VOLUME XI, NUMBER 3 SEPTEMBER/OCTOBER 1989

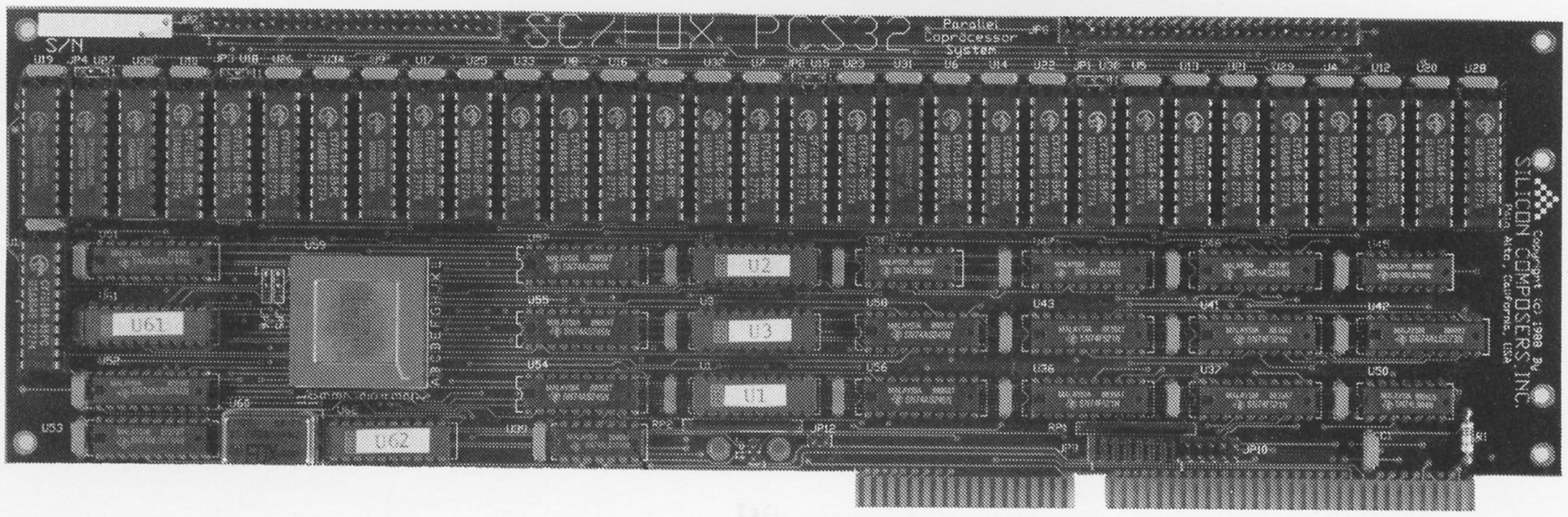
FORTH
DIMENSIONS

MULTIPROCESSOR FORTH

TIME-STATEMENT LEXICON

QUATERNION ROTATIONS

A CHALLENGE OF SORTS



SC/FOX PCS32 (Parallel Coprocessor System32)

FORTH
DIMENSIONS

EDITORIAL

The Forth Interest Group's directors recently decided to distribute public-domain Forth systems on diskette. This is the outgrowth of a years-long debate that has ranged from issues like the support of such software, to the organization's goals, to—not least—the vendor community. In earlier times, some feared that FIG's distribution of public-domain Forth at equally public-domain prices would undercut the market of some Forth vendors. Others saw public-domain consumers as (a) entry-level Forth users who will likely graduate to commercial systems in time, and (b) Forth experts who will use and study systems from any source, but who use commercially supported packages when developing programs for sale or under contract.

Today, most Forth vendors provide complete documentation, technical support, consulting, and custom programming with which FIG-distributed systems will not and cannot compete. FIG does not offer technical support; users who need a supported, comprehensive system must still contact a vendor who can provide one. This issue's "Best of GEnie" discussion will help some prospective users of public-domain systems. It recaps some of the on-line dialog between new and long-time users.

FIG's Mail Order Form is to include F83 v.2.01 and F-PC v.2.25 for IBM PCs and compatibles, and the less-known Pocket Forth for Macintoshes. F83 is familiar to many as the extensive (by comparison with earlier Forths) Forth-83 Model contributed by Henry Laxen and Michael Perry. F-PC is the creation of Tom Zimmer, who has given us a large-environment Forth (files, hypertext, etc., etc.) that deserves lengthy commentary in issues to come. Pocket Forth for the Mac, by Chris Heilman, bills itself as an austere Forth system that follows standard usage and *Starting Forth*, but not rigorously. It can be used to create standalone and desk accessory applications (coming in both forms itself), supports toolbox calls and machine code, and accepts text files.

*   *   *

I want to repeat our call for articles about Forth hardware. Last month's editorial gives details about the closing dates, cash awards, etc. Articles can be about a particular Forth chip or board you have used or built, general design philosophy, a survey of entries in the field, you name it. No one with a vested interest in a product is excluded from writing about it, so long as they honestly admit the affiliation, but our reviewers will look keenly for signs of personal bias in the technical content.

Despite the generally wide fascination with Forth hardware, some of our readers just haven't acquired the taste for it, at least not enough to write about. An event of interest to every programmer is the "Challenge of Sorts" announced in this issue. Dennis Ruffer, head sysop of the GEnie Forth RoundTable, is spearheading this effort to really test your programming skills. The well-designed contest comes complete with prizes and publication for the winners, described later in this issue. The gauntlet has been tossed...

*   *   *

The "Reference Section" continues to grow, with an addition to the on-line resources and two new categories: ANS Forth lists X3J14 representatives who are willing to take your proposals and concerns about the developing ANS Forth directly to that committee; and Forth Instruction provides a place to find ongoing educational resources of interest. Send any additions and elaborations to us.

(Editorial continued on page 37)

Forth Dimensions
Published by the
Forth Interest Group
Volume XI, Number 3
September/October 1989
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Design and Production
Berglund Graphics

Forth Dimensions welcomes editorial material, letters to the editor, and comments from its readers. No responsibility is assumed for accuracy of submissions.

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $30 per year ($42 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices and advertising sales: 408-277-0668.

About the Forth Interest Group

The Forth Interest Group is the association of programmers, managers, and engineers who create practical, Forth-based solutions to real-world needs. Many research hardware and software designs that will advance the general state of the art. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $24/36 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 8231, San Jose, CA 95155."

LETTERS

Object Commentary

Dear Mr. Ouverson,

This is a comment on Mike Elola's comment on my comment on his object-oriented article in *Forth Dimensions* (X/5)!

First, I'm glad that my comment has stirred so much enthusiasm. I have received many positive letters on it. I hope this interest in object-oriented Forth (OOF) continues to grow.

Using the now-famous arithmetic average example, listed below, Elola suggested that the phrase `USE FLOAT` should not be necessary in an OO language.

```
USE FLOAT
A @ B @
+ 2/
C !
```

CSU Forth is *not* an OO language. I found no reason to make it so. Instead, it accommodates OO principles. The original spirit of Forth, as I understood it, is programmer liberty and language extensibility. I therefore designed CSU Forth to be 100% compatible with the standards, yet to offer the programmer an excellent way of program design if he/she wants to use it. In CSU Forth, classes and objects are Forth words. The way they do what they do is found in their `DOES>` parts.

In the arithmetic average example, the phrase `USE FLOAT` is needed for the methods + and `2/`, not for the float objects `A`, `B`, or `C`. These methods will behave differently in a float class than in an integer class. Using `USE`, the programmer doesn't have to create a special object to call these methods. It also allows the programmer to use familiar word names like `2/` without binding ambiguity. The same technique is found in standard Forth whenever you invoke a vocabulary. Perhaps writing the example as below will remove this misunderstanding:

```
A @ B @
USE FLOAT
+ 2/
USE FORTH
C !
```

Finally, I hope that the next Forth standard will *not* be called "Forth++." Forth-90 is what I'd like to see. I especially pray that if the ANSI committee decides to implement OOF, they will *not* be inspired by the cryptic and complex standards of C++. OO principles are much simpler and easier to implement than some folks will lead you to believe.

Sincerely,

Ayman Abu-Mostafa
7932 Lampson Ave. #25
Garden Grove, California 92641-4147

Conditional-Stack Caveat

Dear Marlin,

There is a serious misunderstanding of the `IF ... ELSE ... THEN` and `IF ... THEN` constructs in Abu-Mostafa's article on branchless conditionals ("Forth Needs Three More Stacks,"*FD* XI/1). The standard interpretation of these constructs includes the following points:

1. `IF` removes a single value from the parameter stack and processes it as a Boolean.
2. If the Boolean value is true, processing continues with the words immediately following `IF` and continues up to a matching `ELSE` or `THEN`.

2a. A matching `ELSE` causes a skip of processing of words between `ELSE` and the matching `THEN`. Processing is resumed after `THEN`.

3. If the Boolean value is false, the words immediately following `IF` are skipped up to a matching `ELSE` or `THEN`, and processing is resumed at the matching `ELSE` or `THEN`.
4. The group of words that is skipped has no effect on the parameter stack, the return stack, or any variable in the dictionary, no matter how long or complicated the group of words may be.

In these rules, "matching" means that any `IF ... ELSE ... THEN` or `IF ... THEN` constructs that are nested inside in skipped code must be passed over, and their `ELSE` or `THEN` parts ignored.

One might quibble about whether my wording of the rules is precise, complete, or the most concise possible, but the intent is clear to all users of Forth. In particular, skipped words *must not* affect the parameter stack. Abu-Mostafa's fourth rule of processing his proposed condition stack *does* have skipped `IF` words affecting the parameter stack. This is wrong.

One can implement branchless conditionals correctly without the use of a condition stack. Instead, one needs only an execution control flag (`ECF`) and a nesting depth counter (`NDC`). During initialization of the system, `ECF` is set true and `NDC` is set to zero; then the following rules apply:

1. As each word is parsed from the input stream, ECF is examined. The word is executed if ECF is true, and is skipped if ECF is false.
2. Execution of IF causes the top value on the parameter stack to be stored into ECF.
3. Execution of ELSE causes false to be stored into ECF.
4. Execution of THEN is a no-operation.

During "skipping," special actions are taken if any of the words IF, ELSE, or THEN are encountered, as follows:

1. Skip over IF causes NDC to be incremented.
2. Skip over ELSE causes a test of NDC, and

2a. If NDC is zero, true is stored into ECF.
2b. If NDC is non-zero, no action is taken.

3. Skip over THEN also causes a test of NDC, and

3a. If NDC is zero, true is stored into ECF.
3b. If NDC is non-zero, NDC is decremented.

The use of the NDC makes the "skipping" state pass over matching pairs of IF and THEN words without ending the skipping state. It works up to a nesting level equal to the overflow count of NDC. (Probably larger than the storage allocation of any possible condition stack!) The algorithm is essentially the same as one for finding matching left and right parentheses in algebraic expressions. Evaluation of an algebraic expression is nicely done using a stack to hold intermediate results, but the stack is quite unnecessary if one is only interested in finding matches of parentheses.

In addition to the serious misstatement of IF ... THEN processing, there are other less serious problems with Abu-Mostafa's branchless conditionals:

- Marking words for special processing by setting a bit in the NFA is only satisfactory for interpretive mode. If it is used on compiled code, it would require the outer interpreter to execute >LINK for every word that is being skipped during a skip sequence. This would surely be very slow.
- DO ... LOOP still requires a backward branch. Some of the words that Abu-Mostafa hopes to eliminate with his branchless conditionals will still be required to implement this backward branch.
- The proposed CASE construct requires the programmer to count, by hand, the number of instances of the word CASE and enter that number into the source code explicitly. Surely SELECT and END could be augmented in some way so that they cause the instances of CASE to be counted by the computer rather than by the programmer. People, even programmers, are far less reliable at counting than computers.
- The definition of CASE contains IF. The operation of branchless conditionals requires that the outer interpreter know about the IF that is hidden inside CASE. One bit in the NFA is not enough to encode the necessary information for appropriate processing of CASE. Apparently, the outer interpreter must "open up" the definition of CASE and look inside, find the IF and process it. This must be very slow indeed.

In conclusion, I would like to caution any reader thinking of implementing branchless conditionals on his own Forth to look before he leaps.

Sincerely,

Paul Condon
216 Sheffield Lane
Redwood City, California 94061

Case Counter

In *Forth Dimensions* XI/1, Dr. Ayman Abu-Mostafa suggests adding three stacks to Forth. In addition, he describes a nestable case structure which uses his proposed conditional and case stacks. The scheme for both the stacks and the case statement seems very easy to implement.

The drawback I see to the proposed case structure is that it requires the programmer to specify how many cases are present. Though not a problem when code is initially written, when adding or deleting cases during later maintenance, it will be easy to forget to update the CASES clause. I would like to suggest some minor changes to the case structure that will allow it to count the number of cases for itself.

The change involves keeping the number of cases defined on the case stack underneath the select value. SELECT will initialize this to zero:

```
: SELECT ( n -- )
  0 4 >S
  4 >S  ;
```

(Letters, continued on page 37)

TIME-STATEMENT LEXICON

DAVE EDWARDS - SUBIACO, W.A., AUSTRALIA

Ever since my first process-control program, when I needed to execute code on a timed basis, I have been working on a lexicon for time statements in Forth. Consulting my back copies of *Forth Dimensions*, I discovered a typically excellent article by William Ragsdale (issue V/5) which developed syntaxes of the type:

```
TICK IF EACH-SECOND
TOCK IF EACH-MINUTE
THEN THEN
```

It was perfect for my needs in that first control program. My second program required more sophistication than just each-second and each-minute kinds of statements, so I began extending Ragsdale's basic ideas into the present lexicon.

The lexicon allows a variety of time-based statements in Forth. It does not use an operating-system approach (in which the "system" maintains a set of timers available for use), but constructs in-line Forth code which performs the timing functions. The lexicon allows syntaxes for dealing with a variety of time functions, including:

- A declarable set of time units (milliseconds, seconds, minutes, etc.).
- Perform processes on each *new* time unit (similar to Ragsdale's `TICK TOCK` system).
- Monitor a time *lapse*.
- Declare a time *period.*
- Detect whether a period has *elapsed.*
- Wait to proceed until *after* a specified period.
- Monitor a condition (flag) for a time *period.*

Functions required beyond this set can easily be expressed in phrases using the core lexicon words.

The first design requirement was that the lexicon be able to handle multiple time units: milliseconds, seconds, minutes, even hours and days. To implement this, a set of entities was created that is collectively called the time units, which are simply numbers used by the code to distinguish the current time unit (TU).

Words in the lexicon are designed to run repeatedly inside a Forth control structure for the period of interest. Time lapses are measured by keeping a note of the value of the time units on the previous pass through the loop and comparing it to the value of the time unit on the current pass through the loop. While in the loop, the words need access to various parameters:

> ***The time units are independent of the timing basis.***

- the current time unit (TU)
- the value of the time unit on the last pass (v)
- a counter to accumulate the time lapse (c)
- the specified time period or limit value (P)

All of these parameters are passed on the stack and *they remain on the stack for the duration of the loop.* In this sense, the design uses stack data structures—there is a different data structure for the various aspects of time being monitored:

- To access a time value requires: the time unit (TU).
- To detect a *new* value requires: the time unit and a previous value (v TU).
- To monitor a time *lapse* requires: a counter plus *new*'s parameters (c v TU).
- To detect if a period has *elapsed* requires: the total *period* plus *lapse*'s parameters (P c v TU).

The first word in the lexicon is `@TIME`, which returns the value of the specified time unit. It is used by most words in the lexicon and is shown in Figure One-a. Its definition is implementation-specific and is discussed in the implementation section. It can be used directly to return the value of particular time units, for instance:

```
: @SECS   SECS @TIME ;
: @MINS   MINS @TIME ;
etc.
```

`NEW` expects the previous value of a time unit, along with the time unit, on the stack. This word is analogous to Ragsdale's `TICK` but has been generalized to handle the different units. It leaves an updated TU value and a flag that is true if there was a new value, false otherwise. (See Figure Two.)

Notice that `NEW` actually leaves the difference between the two time unit values on the top of the stack, not a pure flag. This allows `NEW` to be reused in the definition of `LAPSE`.

`NEW` can be used immediately in a Forth control structure to execute `EACH-`*time unit* functions similar to Ragsdale's original system. For example:

```
MINS @TIME
SECS @TIME

  BEGIN
    SECS NEW
      IF EACH-SEC
      THEN SWAP
```

Figure One. Implementation-dependent internals.

Figure One-a

```
: @TIME ( TU -- v )
         CASE
    mSEC OF code to fetch value of milliseconds ENDOF
    SECS OF code to fetch value of Seconds      ENDOF
         etc.                 ENDCASE ;
```

Figure One-b

```
: MAX-TU ( TU -- n )
         CASE
    mSEC OF 1000 ENDOF
    SECS OF 60   ENDOF
    MINS OF 60   ENDOF etc. ENDCASE ;
```

Figure One-c

```
: MAX-TU ( TU -- n )
         CASE
    mSEC OF 10 ENDOF
    cSEC OF 10 ENDOF
    dSEC OF 10 ENDOF
    SECS OF 60 ENDOF etc. ENDCASE ;
```

Figure One-d

```
Physical Timers: VARIABLE msec     contains milliseconds
                 VARIABLE mins     contains minute number

Time-Units:      mSEC  cSEC  dSEC  SECS  MINS  HRS

: @TIME      CASE
    mSEC OF msec @       ENDOF
    cSEC OF msec @ 10 /  ENDOF
    dSEC OF msec @ 100 / ENDOF
    SECS OF secs @       ENDOF
    MINS OF mins @       ENDOF
    HRS  OF mins @ 60 /  ENDOF etc. ENDCASE ;
```

Figure Two. Basic lexicon.

```
: NEW     ( v TU -- v' f \ True if v' not equal to v )
             @TIME DUP ROT - ;

: CLK-ON ( TU -- c=0 v TU \ Clears a LAPSE counter )
             0 SWAP DUP @TIME SWAP ;

: LAPSE ( c v TU -- c' v' TU \ Increments c by time lapse )
             DUP >R NEW DUP
        0< IF R@ MAX-TU + THEN SWAP >R + R> R> ;

: ELAPSED ( P c v TU -- P c' v' TU f \ True if c' >= P )
             LAPSE 3 PICK 3 PICK > ;
```

Figure Three. Control structures.

```
: PERIOD        COMPILE CLK-ON
               [COMPILE] BEGIN
                COMPILE ELAPSED ;         IMMEDIATE

: TIME          COMPILE  OR
               [COMPILE] UNTIL
                COMPILE  2DROP ;          IMMEDIATE
```

```
    MINS NEW
      IF EACH-MIN
      THEN SWAP
  AGAIN
```

Note that current values of the time units are always on the stack during the BEGIN ... UNTIL structure.

CLK-ON sets up the stack values for a time LAPSE monitor—it initializes the lapse count (c) to zero and runs @TIME to provide an initial time unit value. (See Figure Two.)

LAPSE expects a single-precision count, along with NEW's parameters on the stack. The count is incremented by the time lapse between the previous value and the current value, and is shown in Figure Three.

The word MAX-TU in the definition of LAPSE is a word which pushes the maximum value of any TU in the system, and is used to handle wrap-around of the value. For instance, when using SECONDS as the time unit, if the previous value was 59 and the current value is two, then wrap-around has occurred and the MAX-TU for SECONDS (60) must be added to the result left by NEW (-57). MAX-TU is discussed in detail in the implementation section.

LAPSE can, therefore, be run only slightly more frequently than the next higher time unit in the system—if SECS is the current time unit, then LAPSE need only be run once every 59 seconds or so.

LAPSE can be used in a variety of ways. The following phrase leaves a number on the stack which indicates how long it took before the flag-leaving Forth phrase went true:

```
mSEC CLK-ON
BEGIN LAPSE ...  ( f )
UNTIL 2DROP      ( n )
```

The next construct will remain in the loop, running the code between WHILE and REPEAT for ten minutes:

```
MINS CLK-ON
  BEGIN
    LAPSE 3 PICK 10 <
  WHILE ...
  REPEAT
2DROP DROP
```

Optional control structures.

```
: MONITOR     [COMPILE] TIME
                COMPILE 2DROP ;          IMMEDIATE

: DETECTED    [COMPILE] TIME
                COMPILE > ;              IMMEDIATE

: TIMED       [COMPILE] TIME
                COMPILE SWAP
                COMPILE OVER
                COMPILE > ;              IMMEDIATE
```

Note again that care must be taken in the Forth phrases between `WHILE` and `REPEAT`, as there are timing-control numbers on the stack for the duration of the loop.

`ELAPSED` expects a total-period, single-precision number below `LAPSE`'s arguments on the stack. It leaves a flag if the time lapse equals or exceeds the period specified—see Figure Two. As an example of using `ELAPSED`, a common need is for a word which simply consumes the specified period of time. We have named this word `AFTER` and it can be defined as:

```
: AFTER   ( P TU -- )
\ consumes specified time
  CLK-ON
    BEGIN ELAPSED UNTIL
  2DROP 2DROP ;
```

The values used during the loop (period, count, value, time unit) are simply dropped at the end of the loop, being of no further interest in this particular case.

Another common requirement is to wait for a specified total time period and simultaneously monitor some condition. If the condition goes true, the loop is left immediately; otherwise, the total specified period is consumed. This can be accomplished with the phrase:

```
10 MINS CLK-ON
BEGIN ELAPSED ... ( f )
OR UNTIL 2DROP   ( P c )
```

This construct leaves two numbers on the stack, the original limit specified for the period and the lapsed time when the loop terminated.

If the period timed out (the condition did not go true in the specified time period), the two numbers are equal; if the condition went true before the period timed out, the limit is greater than the lapsed time.

The programmer may use these numbers in a variety of ways:

- To continue without testing the termination state, `2DROP` clears the stack.
- To test whether the condition occurred and leave a flag, simply the word > is required.
- To test whether the condition was correctly timed and leave a flag *and the time taken* for the condition to go true, the phrase `SWAP OVER >` is used.

This construct:

```
CLK-ON BEGIN ELAPSED (condition)
OR UNTIL
```

has come to be so useful, and was used so often in process-control code, that compiling words have been defined to build it. The phrase `CLK-ON BEGIN ELAPSED` is built by the word `PERIOD`, and the phrase `OR UNTIL 2DROP` is built by the word `TIME`. Their definitions are given in Figure Three.

This control structure, in conjunction with the subsequent tests, allows statements of the form:

```
10 SECS PERIOD
?HEATED
TIME 2DROP
( to simply proceed )

20 SECS PERIOD
?TERMINAL
TIME > IF
." Key struck within 20 secs"
THEN
```

```
10 MINS PERIOD
?ALARM
TIME SWAP OVER
> IF ." Alarm after "
ELSE ." No alarm within "
THEN . ." minutes"
```

If desired, optional compiling utilities shown in Figure Three can be defined, allowing statements like:

```
20 SECS PERIOD
?TERMINAL DETECTED
IF
." Key struck within 20 secs"
THEN

10 MINS PERIOD
?ALARM  TIMED
IF ." Alarm after "
ELSE ." No alarm within "
THEN . ." minutes"
```

Implementation

The lexicon expects values to which it has access to be changing *automatically* on a timed basis. The physical source of these locations (whether they are created by a hardware interrupt and interrupt code or by a real-time clock) is immaterial to the design.

The steps to implement the timing system are:

1. Decide on the set of time units.
2. Write `@TIME` to interface to the hard ware-specific timing information.
3. Write `MAX-TU` for the set of time units chosen.
4. Use the high-level definitions for the remaining words.

The time units are usually defined in Forth as constants or variables. There is no restriction on the actual values, except that they must be distinct from one another. The interface to the time units is formalized through two words:

`@TIME`	fetches the value of the named time unit.
`MAX-TU`	pushes the maximum (i.e., wrap-around) value of the named time unit.

Note: `MAX-TU` is only used in the word `LAPSE`, so it does not need to be separately defined; it can be written in-line in the definition of `LAPSE`. It is created here simply to clarify the explanation of the required functions.

The structure of `@TIME` is shown in Figure One-a. It uses the time unit on the stack as the input to a `CASE` statement, the clauses of which perform the particular fetch operation and form part of the machine- and even the application-dependent part of the code.

The structure of `MAX-TU` is shown in Figure One-b. It again uses the TU as the input to a `CASE` statement and pushes the wrap-around value of the current time unit *depending on the set of time units chosen in the implementation*, in this case `mSEC`, `SECS`, and `MINS`.

As a further example, if the required set of time units included, say, `mSEC` and hundredths (`cSEC`) and tenths of a second (`dSEC`), then `MAX-TU` would be as shown in Figure One-c.

The chosen set of time units is independent of the physical timing basis. For instance, if a one millisecond interrupt was used to provide variables for milliseconds (msec), seconds (secs), and minutes (mins), it is still possible to implement a set of time units that includes more than just `mSEC`, `SECS`, and `MINS`. This is illustrated in Figure One-d.

Example Implementations

Interrupt-based Timing

Basis of timing: 20 millisecond (50 Hz) interrupt

Variables: `Ticks, Seconds, Minutes, Hours`

Interrupt code:

```
increments Ticks
if Ticks  > 49,
  clear Ticks
  increment Seconds

if Seconds  > 59,
  clear Seconds
  increment Minutes

if Minutes  > 59,
  clear Minutes
  increment Hours
```

In such a system, the allocation of the time units is most conveniently handled by using the `VARIABLE` address as the `CONSTANT` of the time units:

```
Ticks     CONSTANT TICKS
Seconds   CONSTANT SECS
Minutes   CONSTANT MINS
Hours     CONSTANT HRS
```

In effect, no redefinition is needed here—the names of the variables can act as the time units in this particular implementation. Having made this decision, the word `@TIME` is simply Forth's `@` (fetch):

```
: @TIME   @  ;
```

and `MAX-TU` is:

```
: MAX-TU
  DUP TICKS =
    IF DROP 50
    ELSE HRS =
      IF 24 THEN
    THEN  ;
```

Real-time Clock

The Motorola MC 14 6818 Real Time Clock (RTC) has registers which constantly contain the values of:

seconds	(reg 0)
minutes	(reg 2)
hours	(reg 4)

Given these physical addresses, a natural allocation of the time units is:

```
0 CONSTANT SECS
2 CONSTANT MINS
4 CONSTANT HRS
```

A word called `@RTC` is developed to fetch from an RTC register; it expects the register number on the stack and returns the value of that register:

```
: @RTC   ( r -- n )
(code to return the value of )
( Real-Time Clock register r )
;
```

With this definition in place, `@TIME` can be defined as:

```
: @TIME   @RTC  ;
```

and `MAX-TU` can be defined as:

```
: MAX-TU
  DUP HRS =
    IF DROP 24
```

(Continued on page 30)

QUATERNION ROTATION CALCULATION

ANTONIO LARA-FERIA and JOAN VERDAGUER-CODINA
BARCELONA, SPAIN

This program presents the advantage of using quaternions, instead of matrix methods, to calculate rotations. A Forth algorithm to find the unique axis and the angle of a rotation is presented. It is part of a work to apply quaternions in robotics and computer graphics.

According to reference [1], quaternions require fewer mathematical operations than matrix methods. An additional feature of quaternions is that they give the axis and the angle of a rotation directly.

Quaternions can be applied in many areas, for example astronautics [2], mechanics [3], robotics [4,5,6,8], and computer graphics [1,7]. The program presented here focuses only on the use of quaternions to calculate rotations.

Quaternions require fewer math operations.

Numbers and Precision

The version of Forth-83 used is Laboratory Microsystems, Inc. (LMI) PC/Forth 3.10.

The program was written using straightforward, single-length arithmetic. The reasons for doing this are:

- In LMI's version of Forth, no words are provided for multiplying or dividing double-length integers.
- Even though the above-mentioned language can work with an 8087 coprocessor—thus allowing the use of floating-point arithmetic, no 8087 was present on the equipment used to develop this program.

```
Screen # 0
(          ****  QUATERNION  PROGRAM  ****       21:12 04/29/88 )

                ** BY J. VERDAGUER C. **

   (Arranged by GRC)

   -- ACKNOWLEDGEMENTS --

   Written in PC/FORTH V3.1 from Laboratory Microsystems.
   Uses code (SIN & COS routines, plus data tables) from
   LM's utility file FORTH.SCR, which are in screens # 5,
   # 6.

   (C) JVC all but Screens # 5 & # 6.



Screen # 1
(   INITIALISATION & VARIABLE DECLARATION        21:16 04/29/88 )
ASM86 FORTH DEFINITIONS
( Variable declarations )
 3 CONSTANT PI VARIABLE ITER
 VARIABLE VX  VARIABLE VY  VARIABLE VZ
 VARIABLE VGX  VARIABLE VGY  VARIABLE VGZ  VARIABLE MVG
 VARIABLE VGXN  VARIABLE VGYN  VARIABLE VGZN
 VARIABLE AGG
 VARIABLE Q01  VARIABLE Q11  VARIABLE Q21  VARIABLE Q31
 VARIABLE Q02  VARIABLE Q12  VARIABLE Q22  VARIABLE Q32
 VARIABLE Q0T  VARIABLE Q1T  VARIABLE Q2T  VARIABLE Q3T
 VARIABLE A  VARIABLE B  VARIABLE C  VARIABLE D
 VARIABLE R0  VARIABLE R1  VARIABLE R2  VARIABLE R3
 VARIABLE F1



Screen # 2
( VARIABLE SET-TO-ZERO UTILITY '0_INTO'          21:24 04/29/88 )
: 0_INTO
0 VX !  0  VY !  0  VZ !
0  VGX !  0  VGY !  0  VGZ !
0  VGXN !  0  VGYN !  0  VGZN !
0  AGG !
0  Q01 !  0  Q11 !  0  Q21 !  0  Q31 !
0  Q02 !  0  Q12 !  0  Q22 !  0  Q32 !
0  Q0T !  0  Q1T !  0  Q2T !  0  Q3T !
0  A !  0  B !  0  C !  0  D !
0  R0 !  0  R1 !  0  R2 !  0  R3 ! ;
```

```
Screen # 3
( 'X*' & 'X/' SEMI-DOUBLE PRECISION OPER.       21:17 04/29/88 )
: X*
  DROP SWAP DROP * 0 ;
: X/
  DROP SWAP DROP / 0 ;
: DDUP
  OVER OVER ;
: .DTOP
  DDUP .   ;
: SQR
  DUP * ;



Screen # 4
( PQ PROCEDURE                                   21:17 04/29/88 )
 VARIABLE E0   VARIABLE E1   VARIABLE E2   VARIABLE E3
 VARIABLE A0   VARIABLE A1   VARIABLE A2   VARIABLE A3
 VARIABLE Q0   VARIABLE Q1   VARIABLE Q2   VARIABLE Q3
: PQ
  E0 @  A0 @  *  E1 @  A1 @  *  -  E2 @  A2 @  *  -
  E3 @  A3 @  *  -                             Q0 !
  E0 @  A1 @  *  E1 @  A0 @  *  +  E2 @  A3 @  *  +
  E3 @  A2 @  *  -                             Q1 !
  E0 @  A2 @  *  E2 @  A0 @  *  +  E3 @  A1 @  *  +
  E1 @  A3 @  *  -                             Q2 !
  E0 @  A3 @  *  E3 @  A0 @  *  +  E1 @  A2 @  *  +
  E2 @  A1 @  *  -                             Q3 !   ;



Screen # 5
( Sine/cos lookup, from FORTH.SCR file      RGD 17:17 09/20/84 )
( fast SIN and COS by table lookup method, return val*10000 )
FORTH DEFINITIONS DECIMAL

CREATE SINTABLE
0 , 175 , 349 , 523 , 698 , 872 , 1045 , 1219 , 1392 , 1564 ,
1736 , 1908 , 2079 , 2250 , 2419 , 2588 , 2756 , 2924 , 3090 ,
3256 , 3420 , 3584 , 3746 , 3907 , 4067 , 4226 , 4384 , 4540 ,
4695 , 4848 , 5000 , 5150 , 5299 , 5446 , 5592 , 5736 , 5878 ,
6018 , 6157 , 6293 , 6428 , 6561 , 6691 , 6820 , 6947 , 7071 ,
7193 , 7314 , 7431 , 7547 , 7660 , 7771 , 7880 , 7986 , 8090 ,
8192 , 8290 , 8387 , 8480 , 8572 , 8660 , 8746 , 8829 , 8910 ,
8988 , 9063 , 9135 , 9205 , 9272 , 9336 , 9397 , 9455 , 9511 ,
9563 , 9613 , 9659 , 9703 , 9744 , 9781 , 9816 , 9848 , 9877 ,
9903 , 9925 , 9945 , 9962 , 9976 , 9986 , 9994 , 9998 , 10000 ,
-->



Screen # 6
( Sine/cos lookup, from FORTH.SCR file --  RGD 17:18 09/20/84 )
FORTH DEFINITIONS DECIMAL
CREATE TRIG  ASSEMBLER     BX, AX MOV  BX, # 90 CMP   1$ JLE
  BX, # 180 SUB  BX NEG    1$:  BX, 1 SAL
  BX, # SINTABLE ADD  AX, [BX] MOV  RET    FORTH
CREATE SINAX  ASSEMBLER  CWD  BX, # 360 MOV
        BX IDIV  AX, DX MOV  AX, AX OR  2$ JNS   AX, # 360 ADD
   2$:  AX, # 180 CMP  3$ JLE  AX, # 180 SUB
        TRIG CALL  AX NEG  RET
   3$:  TRIG CALL RET    FORTH
CREATE COSAX  ASSEMBLER  AX, # 90 ADD  SINAX JMP  FORTH

( degrees --- cosine )
CODE COS   AX POP  COSAX CALL  AX PUSH  NEXT,  END-CODE
( degrees --- sine )
CODE SIN   AX POP  SINAX CALL  AX PUSH  NEXT,  END-CODE
```

Anyhow, the program can easily be changed to work with floating-point precision simply by entering LMI's Forth editor and changing all single-length arithmetic words to the corresponding floating-point operators.

Overflow and Inexact Results

Since single-length arithmetic has been used, depending on the data the user feeds the program, it may give erroneous results due to internal overflow. Care should be taken to avoid such a situation; sometimes, results shown as the negative of certain values can indicate an internal overflow (e.g., since 32767 is the greatest signed number that can be represented, an overflow-bound sequence like `32767 1+ .` would yield -32768).

On the other hand, when there is no overflow the results may be slightly incorrect due to the poor precision provided by 16-bit signed integer operations.

The magnitude of the two possible errors mentioned above will increase as more and more rotations are performed upon one single vector. In fact, the reasonable maximum number of rotations in such cases turns out to be two.

Extra Code

Some of the words contain code that is not being used by the main RUNME word or the words that it calls. That code expresses programming alternatives; some of the routines and the ideas they represent can be used to change or enhance the program.

How to Run the Program

After entering LMI's PC-Forth, the disk drive containing the screen file QUATERN.SCR should be specified to the system, i.e.:

```
USING <DRIVE>:QUATERN.SCR
```

When the file has been located and acknowledged by PC/Forth, load the program by entering:

```
1 ?SCREENS THRU
```

To execute the program, simply type the word RUNME.

Note: The program uses PC/Forth's assembler in the `SIN` and `COS` routines, so the file ASM86.BIN should be present on the PC/Forth disk. Otherwise, the program won't be loaded.

Bibliography

[1] Lara-Feria A., Verdaguer-Codina J. "Computer Graphics with Quaternions," Seventh International Congress of Cybernetics and Systems. London, September 1987.

[2] Lara-Feria A., Verdaguer-Codina J. "Cuaternios. Aplicación a la Determinación de Actitud de un Satélite," XI Semana Astronaútica. Barcelona, November 1985.

[3] Lara-Feria A., Domingo-Duran J. "Analogía entre la Dinámica del Cuerpo Rígido," XI Semana Astronaútica. Barcelona, November 1985.

[4] Lara-Feria A., Verdaguer-Codina J. "Application de les Quaternions pour Determiner la Position d'un Solide Rigide," Seventh IASTED International Symposium on Robotics and Automation '85. Lugano 1985.

[5] Lara-Feria A., Verdaguer-Codina J. "Applications of Quaternions to Determination of the Rigid Body Position," IFAC Symposium on Robot Control. Barcelona 1985.

[6] Lara-Feria A., Verdaguer-Codina J. "Quaternions Applied to Direct and Inverse Robot Kinematics Problem," IFAC/IFIP/IMACS International Symposium on Theory of Robots. Vienna 1986.

[7] Lara-Feria A., Verdaguer-Codina J. "Teaching Robotics by Simulation," Tenth IASTED International Symposium on Robotics and Automation. Lugano 1987.

[8] Verdaguer-Codina J. "Aplicació de la Cinemàtica Paramètrica al Desenvolupament d'Algoritmes de Control per a Robots Mitjançant Quaternions," E.T.S.E.I.B., Tesi Doctoral. Barcelona 1988.

Joan Verdaguer-Codina works in the Centre d'Alt Rendiment, a high-performance sports center in Catalonia.


```
Screen # 7
( EXTRA WORDS SCREEN-1                                  21:19 04/29/88 )
FORTH DEFINITIONS
: D<>0
  0= NOT
  SWAP 0= NOT
  OR ;
: LLIST  ( initial, final - )
  PRINTER
  1+ SWAP DO
            I LIST LOOP
  CONSOLE ;
: AUTOLOAD
  1 ?SCREENS THRU ;


Screen # 8
( EXTRA WORDS SCREEN-2                                  21:19 04/29/88 )
: 2INPUT
  PAD 1+ 80 EXPECT
  SPAN C@ PAD C!
  PAD 1+ C@ ASCII - = IF
                   0. PAD 1+ CONVERT DROP DNEGATE
                  ELSE
                   0. PAD CONVERT DROP
                  THEN ;
: SEPARATOR CR 80 0 DO ." -" LOOP CR CR ;        : BS 8 EMIT ;
: INPUT
  2INPUT DROP ;
: 2ROLL
  ( Works just like usual ROLL. Remember, the 'top' element )
  ( -'bottom' in HP RPN language- is numbered as the 0th!! )
  1+ DUP ROLL SWAP ROLL ;


Screen # 9
( EXTRA WORDS SCREEN-3                                  21:19 04/29/88 )
: ERROR!
  ." ** WARNING: There may well be (please check) an OVERFLOW "
  ." ERROR in that result **" CR ;
: INFORM1
  ." , all * 10^-" ITER @ . CR
  ERROR! ;
: INFORM2
  ." *10^-" ITER @ 2 * . CR
  ERROR! ;


Screen # 10
( ?PRINTER & MAIN1 SCREEN                               21:19 04/29/88 )
: ?PRINTER
  IF PRINTER ELSE CONSOLE THEN ;
: ASKPRINTER
  ." Wish data to be printed out? (1:Y, 0:N): " INPUT NEGATE
  F1 ! ;
: MAIN1 CR
  1 Q01 !  0 0 0 Q11 !  Q21 !  Q31 !
  CR ." Enter components of vector to be rotated:"
  CR ." X component: " INPUT VX !
  CR ." Y component: " INPUT VY !
  CR ." Z component: " INPUT VZ !
  F1 @ DUP ?PRINTER IF
           SEPARATOR ." Components of vector to be rotated:" CR
           ." X=" VX @  .  CR ." Y=" VY @  .  CR
           ." Z=" VZ @  .  THEN 0 ?PRINTER ;
```

(Screens continued on page 21)

MULTIPROCESSOR FORTH KERNEL

BRADFORD J. RODRIGUEZ - TORONTO, ONTARIO

This article describes a Forth multitasker for a multiple-CPU 68000 system. This multitasker:

- automatically distributes the task load among the available processors, without explicit effort by the programmer;
- provides a means to prevent conflicts when different tasks or different CPUs attempt to use the same resource;
- allows tasks to sit in an idle state, awaiting an external trigger, without polling or other CPU overhead;
- allows interrupts to alter the scheduling of tasks.

The principles described herein can be applied to multiprocessor systems using other CPUs, and even to single-processor systems.

For more throughput, plug in another CPU!

The Application

The multiprocessor kernel was originally developed for a performance-lighting control system. The processing demands of this system were quite strict, and fell into three categories:

- Event-driven processing—initiated by external events, such as the system operator moving a control handle. Requires a response time on the order of 100 milliseconds (msec).
- Time-driven processing—must occur at periodic intervals. Most of this repetitive processing occurs every 40 msec, but intervals from ten msec to 1000 seconds

Listing One

```
                                                       Scr #   180
\ **************  MULTIPROCESSOR TASKER v3  04 02 86 BJR *******
4 CONSTANT CELL
: CELLS ( n - n)   4 * ;

: SUBROUTINE   0 VARIABLE -4 ALLOT  [COMPILE] ASSEMBLER ; \ Fig
\ : SUBROUTINE   CREATE  [COMPILE] ASSEMBLER ;        \ Forth-83

181 LOAD       \ task area definition
182 LOAD       \ internal data areas
183 LOAD       \ tasker subroutines
189 LOAD       \ tasker primitives
190 LOAD       \ defining words & initialization
191 LOAD       \ task setup



                                                       Scr #   181
\ Task area structure                         11 02 86 BJR
\ Offsets into the task area
HEX 80 CELLS CONSTANT USIZE    \ size of user variables
0    CONSTANT UAREA    USIZE +     \ user variables
DUP CONSTANT RSTACK  80 CELLS +   \ return stack
DUP CONSTANT RTOP
DUP CONSTANT PSTACK  80 CELLS +    \ parameter stack
DUP CONSTANT PTOP     4 CELLS +    \ top safety margin
    CONSTANT TASKSIZE       \ total size of the task area
DECIMAL
\ offsets (from UAREA) to selected user variables
 2 CELLS CONSTANT +RP-TEMP      3 CELLS CONSTANT +S0
 4 CELLS CONSTANT +R0           5 CELLS CONSTANT +TIB

: TASK   0 VARIABLE  TASKSIZE CELL - ALLOT ;   \ Fig
\ : TASK   CREATE  TASKSIZE ALLOT ;            \ Forth-83

                                                       Scr #   182
\ Semaphore queues structure               ( 30  7 86 BJR 15:30 )
DECIMAL
8 CONSTANT .SEMA  10 CONSTANT .IBIT    \ semaphore field offsets

: SEMAPHORE   0 VARIABLE  2 CELLS ALLOT ;   \ Fig
\ : SEMAPHORE   CREATE  3 CELLS ALLOT ;     \ Forth-83

SEMAPHORE READYQ                     \ ready queue header, 3 cells

HEX C8 USER SELFQ                    \ "self-queue" to suspend tasks
\ 3 cells in task's user area; actual offset is system dependent

0 USER MYTASK        \ returns addr of currently-executing task!
SELFQ MYTASK - CONSTANT +SELFQ  \ offset from base of task area
;S

```

```
                                                  Scr #   183
\ Tasker -- (start)                         (68000( 21  7 86 BJR 23:45 )
SUBROUTINE (START) HEX    \ ar1 = new task
\ Put given task on ready queue
   SR DR7   .W MOV,                       \ save interrupt level
   READYQ #L AR0 .L MOV,
   700 # SR .W OR,  .IBIT AR0 &[ .B TAS,  MI HERE 4- *+ BCC,
   1 # .SEMA AR0 &[ .W SUBQ,                        \ decrement semaphor
   ( GE IF,  impossible condition: readyq less than empty )
   4 AR0 &[ AR2 .L MOV,  AR1 AR2 [ .L MOV,  \ put task on queue
   AR1 4 AR0 &[ .L MOV,  AR0 AR1 [ .L MOV,
   7 # .IBIT AR0 &[ .B BCLR,                \ release queue
\ Resume execution
   DR7   SR .W MOV,   RTS,  ;C     \ restore interrupt level
-->
```

```
                                                  Scr #   184
\   Tasker -- (signal)                      (68000( 21  7 86 BJR 23:45 )
SUBROUTINE (SIGNAL) HEX  \ ar0 = semaphore adr, u = current task
\ Increment semaphore count
   SR DR7   .W MOV,                       \ save interrupt level
   700 # SR .W OR,  .IBIT AR0 &[ .B TAS,  MI HERE 4- *+ BCC,
   1 # .SEMA AR0 &[ .W ADDQ,                        \ increment semaphor
\ If count > 0, continue
   GT IF,  7 # .IBIT AR0 &[ .B BCLR, ELSE,  \ continue,divisible
\ If count <= 0, get task from semaphore queue
   IP RP -[ .L MOV,  S RP -[ .L MOV,  RP 8 U &[ .L MOV,  \ save
   AR0 [ AR1 .L MOV, AR1 [ AR0 .L CMP,       \ get head of sema q
   EQ IF,  AR0 4 AR0 &[ .L MOV,  THEN,  AR1 [ AR0 [ .L MOV,
   7 # .IBIT AR0 &[ .B BCLR,                \ release queue
-->
```

```
                                                  Scr #   185
\   Tasker -- (signal)                      (68000( 21  7 86 BJR 23:45 )
\     put current task on ready queue
   READYQ #L AR0 .L MOV,
   700 # SR .W OR,  .IBIT AR0 &[ .B TAS,  MI HERE 4- *+ BCC,
   1 # .SEMA AR0 &[ .W SUBQ,                        \ decrement semaphor
   ( GE IF,  impossible condition: readyq less than empty )
   4 AR0 &[ AR2 .L MOV,  U   AR2 [ .L MOV,  \ put cur on ready q
   U   4 AR0 &[ .L MOV,  AR0 U   [ .L MOV,
   7 # .IBIT AR0 &[ .B BCLR,                \ release queue
\     make semaphore's task current
   AR1 U .L MOV,                            \ switch to new task
   8 U &[ RP .L MOV,  RP [+ S .L MOV,  RP [+ IP .L MOV,  \ restr
\ Resume execution
   THEN,  DR7   SR .W MOV,   RTS,  ;C  \ restore interrupt level
-->               uses dr7,ar1,ar2  expects u,sp,rp valid
```

```
                                                  Scr #   186
\   Tasker -- (wait)                        (68000( 21  7 86 BJR 23:45 )
SUBROUTINE (WAIT)        \ ar0 = semaphore adr, u = current task
\ Decrement semaphore count
   SR DR7   .W MOV,                       \ save interrupt level
   700 # SR .W OR,  .IBIT AR0 &[ .B TAS,  MI HERE 4- *+ BCC, \ i
   1 # .SEMA AR0 &[ .W SUBQ,                        \ decrement semaphor
\ If count >= 0, continue
   GE IF,  7 # .IBIT AR0 &[ .B BCLR, ELSE,  \ continue,divisible
\ If count < 0, put task on semaphore queue
   IP RP -[ .L MOV,  S RP -[ .L MOV,  RP 8 U &[ .L MOV,  \ save
   4 AR0 &[ AR2 .L MOV,  U   AR2 [ .L MOV,  \ put cur on sema q
   U   4 AR0 &[ .L MOV,  AR0 U   [ .L MOV,
   7 # .IBIT AR0 &[ .B BCLR,                \ release queue
-->
```

are possible.

- Background processing—continuous processes, such as display updates, which have relaxed timing requirements.

Systems whose capacity varied over a ten-to-one range were to be sold. So that the processing power could be configured to suit, the architecture was designed (Figure One) to use from one to four 68000 processors, with a common memory of CMOS RAM. The processors each had private EPROMs for program storage.

The programmer needn't know how many CPUs are installed.

Requirements of the Multitasker

The 68000s were to be programmed in multitasking Forth. Our original software design assumed a round-robin tasker; however, we soon switched to a more "traditional" queued tasker for several reasons:

1. We expected to have a large number of idle tasks—awaiting some external or timed event—at all times. Our studies *for this configuration* indicated that, if 80% or more of the tasks were idle, the queued tasker is more efficient. (If fewer than 80% are idle, it is faster to poll them round-robin than to move them on and off a "ready queue.")

2. We needed to guarantee *order of service*. Our resources—particularly communications—required that competing requests be serviced on a first-come, first served basis.

3. Curiosity. We hadn't seen a queued tasker in Forth, and wanted to see how easily it could be done!

Queue Storage

Figure Two shows the structure of a task queue. Each queue is stored as a singly linked list. The link is stored in the first cell of each task's user area, looking very much like a typical round-robin multitasker. The differences are:

- there are *many* linked lists,
- they are *not* linked in a circle, and

```
                                                    Scr #   187
\   Tasker -- (wait)                     (68000( 21  7 86 BJR 23:45 )
\    get current task from ready queue
   READYQ #L AR0 .L MOV,
   700 # SR .W OR,  .IBIT AR0 &[ .B TAS,  MI HERE 4- *+ BCC, \ i
   1 # .SEMA AR0 &[ .W ADDQ,                 \ increment semaphor
   GT IF,  0 # TRAP,  ( ready queue empty! )  THEN,
   AR0 [   U .L MOV,   U [ AR0 .L CMP,       \ get head of readyq
   EQ IF,  AR0 4 AR0 &[ .L MOV,  THEN,   U [ AR0 [ .L MOV,
   7 # .IBIT AR0 &[ .B BCLR,                  \ release queue
\    make semaphore's task current
   8 U &[ RP .L MOV,  RP [+ S .L MOV,  RP [+ IP .L MOV,  \ restr
\ Resume execution
   THEN,  DR7   SR .W MOV,   RTS,  ;C  \ restore interrupt level
-->             uses dr7,ar2  expects u,sp,rp valid
```

```
                                                    Scr #   188
\   Tasker -- (next)                     (68000) 12 02 86 BJR
SUBROUTINE (NEXT)    NEXT ;C      \ start Forth inner interpreter
DECIMAL ;S

(SIGNAL) and (WAIT) assume that all context has been stacked,
and that the last thing stacked is the PC for the restore.
This is normally accomplished by entering via JSR.

This subroutine is made the starting PC of a newly-initialized
task.  When the new task is started from a queue, its IP and SP
will be unstacked, and then (NEXT) will be entered...starting
high-level execution at the given IP.
```

```
                                                    Scr #   189
\ Tasker -- start - pause                (68000( 30  7 86 BJR 17:32 )
CODE START     S [+ AR1 .L MOV,          \ tadr -- ! start new task
   (START)   *+ BSR,    NEXT  ;C
CODE SIGNAL    S [+ AR0 .L MOV,          \ qadr -- ! release resorce
   (SIGNAL) *+ BSR,    NEXT  ;C
CODE WAIT      S [+ AR0 .L MOV,          \ qadr -- ! acquire resorce
   (WAIT)    *+ BSR,    NEXT  ;C
CODE PAUSE     READYQ #L AR0 .L MOV,   \ -- ! switch to next task
   (SIGNAL) *+ BSR,    NEXT  ;C
CODE SUSPEND  SELFQ MYTASK - U [ AR0 LEA, \ -- ! suspend self
   (WAIT)    *+ BSR,    NEXT  ;C
CODE RESUME    S [+ AR0 .L MOV,         \ taskadr -- ! resume task
   SELFQ MYTASK - #L AR0 ADD,  (SIGNAL) *+ BSR,   NEXT  ;C
;S
Note that the Forth context information (S,IP,RP) is saved by
the task switching primitives.
```

```
                                                    Scr #   190
\ Tasker -- newdevice - newresource        ( 30  7 86 BJR 15:43 )
HEX
: NEWDEVICE     \ qadr -- ! initialize semaphore to 0 for event
   DUP DUP !  DUP DUP CELL + !  0 SWAP 2 CELLS + ! ;

: NEWRESOURCE  \ qadr -- ! init. semaphore to 1 for shared resou
   DUP NEWDEVICE  10000 SWAP 2 CELLS + ! ;

DECIMAL ;S
```

- tasks are constantly being moved (i.e., relinked) from one list to another—the task order is dynamic, rather than static.

This approach involves a minimum of data movement. A task can be moved from one queue to another by changing four links.

The linked list requires a very small memory overhead—three cells (12 bytes) per queue. This "queue header" contains a head pointer, a tail pointer, a 16-bit integer semaphore, and a multiprocessor "lock" bit.

A snapshot of the queues during execution might look something like Figure Three.

Allocation of Tasks

Tasks which are ready to run are held on a ready queue. When a CPU finishes one task—perhaps by executing `PAUSE`—it will pick up the next task from the head of the ready queue.

All the CPUs pick up tasks from the same ready queue, so the first CPU to become available will service the first waiting task. Since all CPUs see the same memory and I/O space, and have identical copies of the program, *any* task can run on *any* CPU.

This means that the programmer does not need to know which CPU his code is to run on. In fact, the programmer does not need to know how many CPUs are installed. The task load is automatically divided among the installed CPUs. For more throughput, plug in another processor!

The limiting factor is bus contention. We minimized this by giving each CPU a private (but identical) program memory, but still the VME bus becomes saturated when three or four CPUs compete for data memory.

Protection of Shared Resources

Our critical resources were protected against conflicting access with the classic "semaphore" operators, `WAIT` and `SIGNAL`. Most textbooks on operating systems describe these in detail, so this will be just an overview.

Each protected resource has an integer semaphore. Its initial value, +1, indicates that the resource is available. A zero semaphore means the resource is in use. A negative value, -N, indicates that it is in use and that there are N pending requests for the resource.

Whenever a task requests a busy resource, it is placed on a "wait queue" for that resource. This queue is first-in, first-out—or, more to the point, first-come, first-served. Tasks on a wait queue consume no CPU time.

The programmer does this through the operators `WAIT` and `SIGNAL`.

`WAIT` decrements a given semaphore. If the resource is busy, the task is parked on the wait queue, and a new task is started from the ready queue (Figure Four).

`SIGNAL` increments the semaphore. If a task is waiting for this resource, pause the current task and start the waiting task (Figure Five).

`WAIT` and `SIGNAL` surround the code which uses the protected resource, as follows:

```
SEMAPHORE DISK
: xxx  DISK WAIT
  code to access disk
  DISK SIGNAL  ;
```

Note that we can always tell the state of the resource and its wait queue by examining the semaphore value. In our implementation, the semaphore and the header for the wait queue are stored together (Figure Two).

(For those familiar with the "monitor" construct used in many concurrent languages: monitors can be implemented very easily with semaphores. Each monitor requires one semaphore, and all routines in the monitor `WAIT` and `SIGNAL` that semaphore.)

`WAIT` and `SIGNAL` are required to be *indivisible*. Nothing must alter or use the semaphore and queue data structure while a `WAIT` or `SIGNAL` is in progress. In a single-CPU system, this is done by disabling interrupts. In a multiple-CPU system, we must further guard against, say, two processors `WAIT`ing the same semaphore simultaneously.

In the 68000, this is done with the indivisible TAS (Test And Set) instruction. This instruction is not powerful enough to use in place of semaphores, but it is sufficient to protect the semaphores themselves from conflicting access. Figure Six shows

```
                                                           Scr #   191
\ Tasker -- init-task                             ( 30  7 86 BJR 15:38 )
RTOP 3 CELLS - CONSTANT RINIT    \ initially 3 cells stacked

: INIT-TASK   \ init-ip taskadr -- ! word to set up new task area
   MYTASK OVER UAREA + USIZE CMOVE \ copy user vars from MYTASK
   DUP RSTACK + OVER +TIB + !      \ top of user area -> TIB
   DUP RTOP +   OVER +R0 +  !      \ top of rtn stack -> R0
   DUP RINIT +  OVER +RP-TEMP + !  \ 3 pushes down -> RP-TEMP
   DUP PTOP +   OVER +S0 +  !      \ top of param stack -> S0
   DUP >R PTOP + (NEXT) ROT ROT    \ pc,init-ip,sp: task context
   SP@  R> RINIT + 3 CELLS CMOVE   \ copy to task return stack
   2DROP DROP ;

;S  This word assumes that execution of the new task is to begin
with a high-level Forth word (as specified by init-ip).


                                                           Scr #   192
\ Tasker -- nulltask - coldstart                  ( 30  7 86 BJR 15:43 )
TASK NULLTASK                         \ "do-nothing" task for readyq

: DONULL  BEGIN PAUSE AGAIN ;     \ must be a hi-level word!

: COLDSTART   READYQ NEWDEVICE    \ initially have 0 tasks on q
   SELFQ NEWDEVICE                \ init. selfq (to be spawned)
    ' DONULL NULLTASK INIT-TASK  ;        \ Fig
\   ' DONULL >BODY NULLTASK INIT-TASK ;   \ Forth-83

DECIMAL ;S


                                                           Scr #   193
\ Multiprocessor tasker glossary                  05 02 86 BJR
WAIT              qptr --
(WAIT)            ar0 = qptr
        Wait on indicated semaphore.  Semaphore is decremented.
        If "available", execution proceeds.  If "busy", the task
        is placed on the semaphore queue, and a task is started
        from the ready queue.

SIGNAL            qptr --
(SIGNAL)          ar0 = qptr
        Signal indicated semaphore.   Semaphore is incremented.
        If then "available", execution proceeds.  If a task is
        waiting on the semaphore, the current task is put on the
        ready queue and the waiting task is started.


                                                           Scr #   194
\ Multiprocessor tasker glossary                  11 02 86 BJR
START             taskadr --
(START)           ar1 = taskadr
        Put the given task on the ready queue.

PAUSE             --
        Suspend current task, and start next available task in
        the ready queue.

SUSPEND           --
        Current task is suspended and put on the its internal
        "self-queue".  The next ready task is started.

RESUME            taskadr --
        Suspend current task, and start execution of the given
        task if it was SUSPENDed.
```

how TAS is used to make the semaphore operations indivisible. (This lock-and-unlock action is also shown in Figures Four and Five.)

Note that the "busy bit" in the semaphore only means that the *semaphore* is busy, not that the resource is busy. So that the CPUs don't spend time waiting on this bit, we ensure that the only routines which set this bit also clear it after a few dozen instructions at most.

Managing Interrupts with `WAIT` and `SIGNAL`

We wish to be able to start a task on the occurrence of an interrupt. Presumably, this task will have been in an idle state, waiting for the interrupt. `WAIT` and `SIGNAL` let us do this. A semaphore (and queue) are defined for an interrupt, with the difference that the semaphore is initialized to zero instead of +1. The interrupt service task then `WAIT`s on this semaphore, causing it to be parked on the wait queue.

Some other task will be running when the interrupt occurs. The interrupt handler saves *all* of the machine context on that task's stack, then calls `(SIGNAL)`. `(SIGNAL)` stacks the PC, puts the running task on the ready queue, and starts the service task which was waiting on the semaphore queue.

When the service task completes, it will `WAIT` again. Eventually, the task that was interrupted will be pulled from the ready queue and its PC popped from its stack. The PC that was stacked points into the interrupt handler code, just after the call to `(SIGNAL)`. This will be the code to restore the full context and return from interrupt. (Figure Seven shows how the context is stored by `WAIT` and `SIGNAL`.)

The task is always resumed at the point in the machine code where it was suspended. This allows a different context to be saved for programmer and interrupt-driven task switches. A high-level task switch (e.g., `PAUSE`) need only save `IP`, `RP`, and `SP`.

The Listing

Listing One is the 68000 assembler code for the tasker. It was written in a fig-FORTH derivative, so there are some differences from the Forth-83 Standard.

`SUBROUTINE` defines a code word which simply returns its address when executed. The notation used here is `(xxx)`

```
                                                          Scr #    195
\ Multiprocessor tasker glossary
SEMAPHORE name    ---
        Allocate space for a semaphore and queue header, and
        define "name" to return its address when executed.

TASK name         ---
        Allocate space for a task (user area and stacks), and
        define "name" to return its address when executed.

INIT-TASK        init-ip taskadr ---
        Initialize the user variables and stack pointers for the
        given task, and save its machine context so that it will
        begin interpretation at init-ip when activated.
        init-ip must point to high-level Forth code.
        INIT-TASK must be used before STARTing a task.
```

```
                                                          Scr #    196
\ Multiprocessor tasker glossary
NEWDEVICE        qadr ---
        Set the semaphore to 0 and its queue header to "empty."
        Used to initialize a semaphore for an interrupt.

NEWRESOURCE      qadr ---
        Set the semaphore to +1 and its queue header to "empty."
        Used to initialize asemaphore for a shared resource.
```

(*Figures continued on next page*)

(Figures continued from previous page)

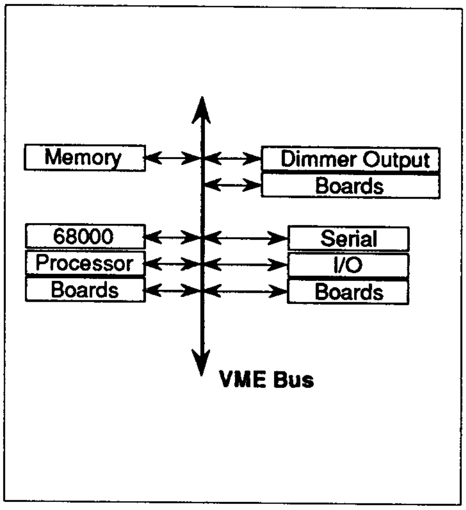


Figure One. The hardware.

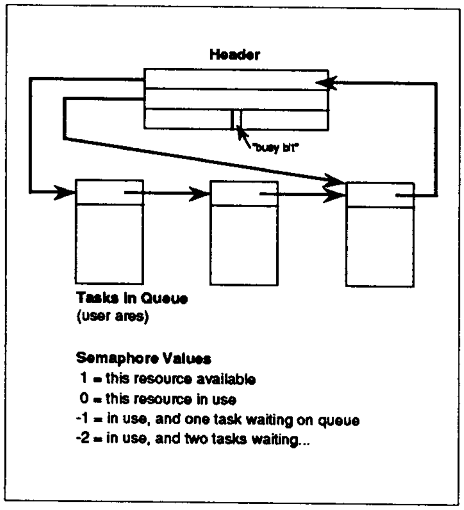


Figure Two. A queue.

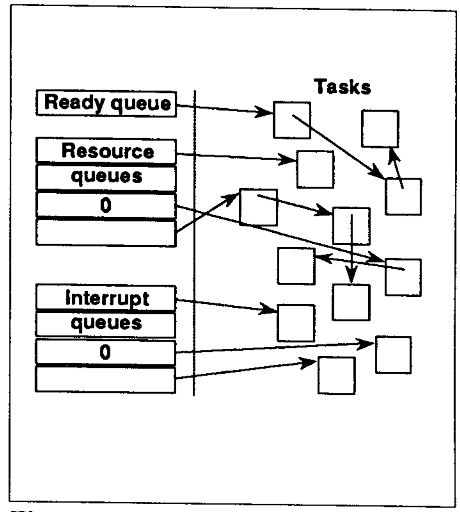


Figure Three. System queues.

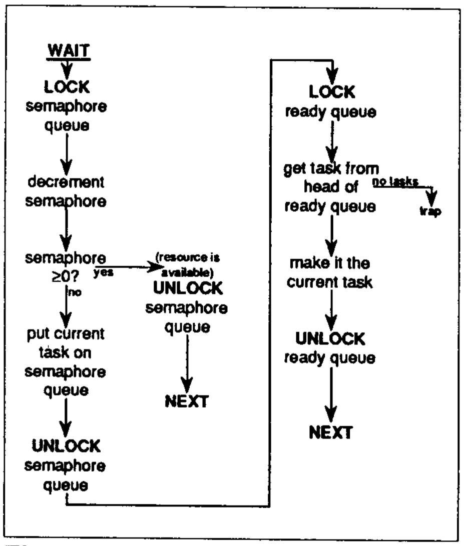


Figure Four. `WAIT`.

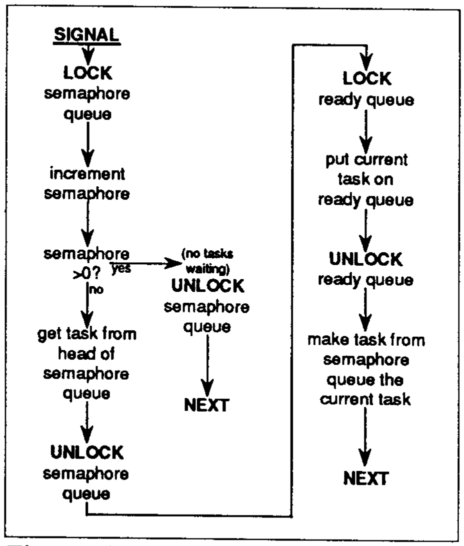


Figure Five. `SIGNAL`.

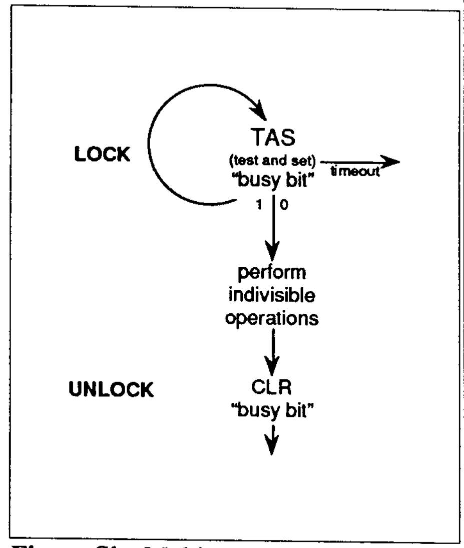


Figure Six. Multiprocessor protection.

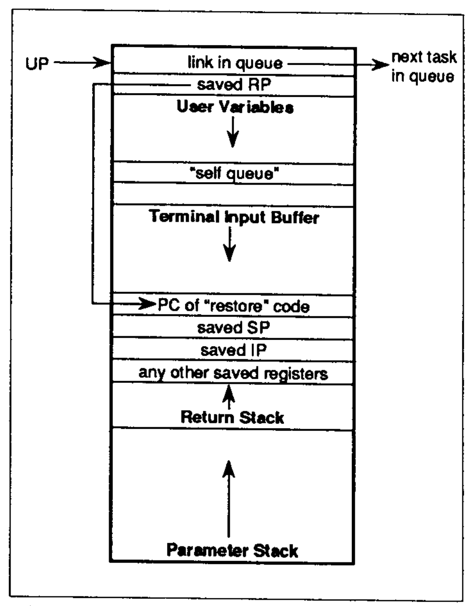


Figure Seven. A task's user area.

Advertisers Index

for an assembly language subroutine; xxx for the executable Forth word.

Screen 181 defines the layout of the task area (Figure Seven). TASK allocates this space in a named data structure.

Screen 182 defines some of the layout of a semaphore queue header. SEMAPHORE is the defining word. Note that the ready queue is defined the same as a semaphore queue.

Every task includes a "private" queue header in its user variables area. This "self queue" is used by SUSPEND and RESUME (described below).

(START) activates a task for the first time by putting it on the tail of the ready queue (see screen 183). From this moment on, except when executing, the task will always be on some queue or other. The remaining words simply move tasks from queue to queue.

(SIGNAL) and (WAIT) are the basic task-control routines. They are callable from a machine-language routine, such as an interrupt handler.

The Forth-callable START, SIGNAL, and WAIT are on screen 189.

Note that a PAUSE (voluntary task switch) is achieved by simply SIGNALing the ready queue. (Follow the logic in Figure Five.)

A task is SUSPENDed by causing it to WAIT on its self queue, whose semaphore is initialized to zero. Another task can SIGNAL that semaphore to RESUME the suspended task.

NEWDEVICE and NEWRESOURCE initialize a semaphore queue for an interrupt and a shared resource, respectively.

INIT-TASK initializes a task area created by TASK. It stacks a context such that the task will begin high-level execution at init-ip (which should be the parameter field address of a colon definition).

Screen 192 illustrates the creation of a task, a do-nothing task in this case. Defining one such task per CPU will ensure that the ready queue is never empty (an error condition).

COLDSTART shows how the multitasker, boot task, and defined tasks are initialized in a colon definition. Some such word will be required in the final system's startup code.

Where to Go From Here

This implementation was adequate for our needs, but it can certainly be taken

(Screens from page 13)

```
Screen # 11
( MAIN2, MAIN3 SCREEN                              21:20 04/29/88 )
: MAIN2 CR
  F1 @ DUP ?PRINTER IF
                   CR ." Vector to be rotated:" CR
                   ." (" VX @  .  BS ." ," VY @  .  BS ." ,"
                   VZ @  .  BS ." )" CR THEN 0 ?PRINTER ;
: MAIN3 CR
  CR ." Enter components of Quaternion Axis:" CR
  ." X component: " INPUT VGX !  CR
  VGX @  0=  IF
      ." Y component: " INPUT CR DUP  VGY ! 0=  IF
                         ." Z component: " INPUT CR VGZ !
     ELSE 0 VGZ ! THEN
  ELSE 0 DUP  VGY !  VGZ ! THEN ;


Screen # 12
( MAIN4 SCREEN                                     21:20 04/29/88 )
: MAIN4 CR
  ." Angle to rotate (in degrees): " INPUT AGG !
  F1 @ DUP ?PRINTER IF
                   CR ." Quaternion rotation vector:" CR
                   ." X component: " VGX @  .  CR
                   ." Y component: " VGY @  .  CR
                   ." Z component: " VGZ @  .  CR
              " Quaternion Gyration Axis: (" VGX @    BS ." ,"
                   VGY @  .  BS ." ," VGZ @  .  BS ." )" CR
                   ." Angle to rotate (in degrees): "
                   AGG @  .  CR THEN 0 ?PRINTER
  VGX @  SQR VGY @  SQR +  VGZ @  SQR + 0 2SQRT MVG !
  VGX @  MVG @  /  VGXN !       VGY @  MVG @  /       VGYN !
  VGZ @  MVG @  /       VGZN !  ;


Screen # 13
( MAIN5 SCREEN                                     21:20 04/29/88 )
: MAIN5 CR
  AGG  @ 2 /  DUP  DUP  DUP
                        COS 1000 / Q02 !
                        SIN 1000 / VGXN @  *  Q12 !
                        SIN 1000 / VGYN @  *  Q22 !
                        SIN 1000 / VGZN @  *  Q32 !
  ( Transfer of values to those in PQ word )
  Q02 @  Q12 @  Q22 @  Q32 @
  Q01 @  Q11 @  Q21 @  Q31 @
  A3 !  A2 !  A1 !  A0 !
  E3 !  E2 !  E1 !  E0 !
  PQ
  Q0 @  Q1 @  Q2 @  Q3 @
  Q3T !  Q2T !  Q1T !  Q0T !
  Q0T @  Q01 !  Q1T @  Q11 !  Q2T @  Q21 !  Q3T @  Q31 !  ;


Screen # 14
( CONVERT-TO-NORMAL utilities                      21:20 04/29/88 )
( For coping with the inexistence of 8087 co-processor... )
: CONVERT-TO-NORMAL
  Q0T @  Q1T @  Q2T @  Q3T @
  R1 @  R2 @  R3 @
  7 0 DO
      6 ROLL 10000 /
     LOOP
  R3 !  R2 !  R1 !
  Q3T !  Q2T !  Q1T !  Q0T !  ;
: /1E4 ITER @ 0 DO 10000 /  LOOP ;
: /1E2 ITER @ 0 DO 100 /  LOOP ;
: DIVIDE
  R3 @ /1E4 R3 !  R2 @ /1E4 R2 ! R1 @ /1E4
  R1 ! Q0T @ /1E2 Q0T !  Q1T @  /1E2 Q1T !
  Q2T @ /1E2 Q2T !  Q3T @  /1E2 Q3T !  ;
```

(Screens continued on next page)

```
Screen # 15
( ASK_CONT1, MAIN6                                   21:21 04/29/88 )
: ASK_CONT1
  ." Want more than one turn for the same vector? (1:Y, 0:N): "
  INPUT NEGATE ;
: MAIN6 CR
  0 VX @  VY @  VZ @  Q0T @  Q1T @ NEGATE Q2T @ NEGATE
  Q3T @ NEGATE
  A3 !  A2 !  A1 !  A0 !
  E3 !  E2 !  E1 !  E0 !          PQ
  Q0 @  Q1 @  Q2 @  Q3 @
  D !  C !  B !  A !
  Q0T @  Q1T @  Q2T @  Q3T @  A @  B @  C @  D @
  A3 !  A2 !  A1 !  A0 !
  E3 !  E2 !  E1 !  E0 !          PQ
  Q0 @  Q1 @  Q2 @  Q3 @
  R3 !  R2 !  R1 !  R0 !  ;




Screen # 16
( MAIN7, ASK_CONT2                                   21:21 04/29/88 )
: MAIN7 CR
  F1 @ DUP ?PRINTER IF
       CR ." Total rotation by Quaternions is:"
       CR ." Q=(" Q0T @  .  BS ." )e0+(" Q1T @  .  BS ." )e1+("
          Q2T @  .  BS ." )e2+(" Q3T @  .  BS ." )e3" INFORM1
       CR ." The resultant rotated vector is:"
       CR ." R=(" R1 @  .  BS ." ," R2 @  .  BS ." ,"
       R3 @  .  BS ." )" INFORM2 CR CR
  THEN 0 ?PRINTER
  CR CR ." Total rotation by Quaternions is:"
    CR ." Q=(" Q0T @  .  BS ." )e0+(" Q1T @  .  BS ." )e1+("
    Q2T @  .  BS ." )e2+(" Q3T @  .  BS ." )e3" INFORM1
    CR ." The resultant rotated vector is:"  CR ." R=("
    R1 @ . BS ." ," R2 @ . BS ." ," R3 @ . BS ." )" INFORM2 ;
 : ASK_CONT2 ." Enter 1 to continue, 0 to stop: " INPUT NEGATE ;




Screen # 17
( MAIN PROGRAM: RUNME                                21:21 04/29/88 )
: RUNME
  VINIT SEPARATOR
  ."                     *** FORTH QUATERNION PROGRAM ***" CR
  SEPARATOR CR ASKPRINTER
  BEGIN
       0_INTO MAIN1 MAIN2
       0 ITER !
       BEGIN
            MAIN3 MAIN4 MAIN5
            ITER @ 1+ ITER !
            CR ASK_CONT1 NOT
       UNTIL
       MAIN6 ( DIVIDE )
       MAIN7 CR ASK_CONT2 NOT
  UNTIL ;
```

further.

Support could be included for private tasks, i.e., tasks restricted to one CPU and to that CPU's memory. This would largely solve the problem of bus saturation.

We have prototyped a round-robin tasker with multiprocessor support; this may be better suited to many applications.

Finally, the principles of the 68000 multiprocessor tasker can be applied to other CPUs!

References

Humbert-Droz and Jansson, *McPascal*, Algotech Computer Corporation, 1980. Description of monitors used in Micro-Concurrent Pascal.

Knuth, *The Art of Computer Programming*, Volume One: "Fundamental Algorithms," Addison-Wesley, 1968. For everything you ever wanted to know about linked lists.

Madnick and Donovan, *Operating Systems*, McGraw-Hill Computer Science Series.

Tsichritzis and Bernstein, *Operating Systems*, Academic Press, 1974. Description of semaphores on pp. 34–38.

Bradford J. Rodriguez is a freelance software/hardware designer specializing in real-time control applications. He discovered Forth as a student in 1978, but only recently was seduced into speaking and writing about it.

SEARCH ORDER STRUCTURE

CHESTER H. PAGE - SILVER SPRING, MARYLAND

I have developed a simple vocabulary search-order routine in which
`VOC1 SEARCHES`
`VOC2 SEARCHES`
`VOC3`
establishes the specified search order, with `VOC3` and all other vocabularies followed immediately by `FORTH`. Entering `NORMAL.SEARCH` restores the default condition of each vocabulary being followed by `FORTH` in the search order.

My routine is based on a vocabulary name structure using the dummy link as a pointer to the last word in the vocabulary, and a dummy parameter as a pointer to the dummy link of the next vocabulary to be searched. (See Figure One.)

A departure from tradition.

`CONTEXT/CURRENT` point to the appropriate dummy link, which in turn points to the last word in <vname>. *The first word in <vname> has its link point to the dummy name (81A0) in <vname>.* Thus, in an empty vocabulary, the "last word" is the dummy name of that vocabulary. See Figure Two for `<vname2>`. This is a departure from tradition. The dummy parameter in `<vname>` points to the dummy link of the next vocabulary in the search order, normally `FORTH`.

Words needed for constructing and searching vocabularies: A five-parameter variable `VOC.LIST` holding the names of all vocabularies; and a system variable `SEARCH.VOC` playing the role of `CONTEXT`, pointing to the top-word pointer of the vocabulary to be searched.

```
: SET.CONTEXT 2+ CONTEXT ! ;

: VOCABULARY FORTH DEFINITIONS CREATE 2 ALLOT [ BASE @ HEX ]
A081 ,  [ BASE ! ] HERE 2- , ['] FORTH 6 + ,
         \ Build vname
 LATEST 2 BEGIN DUP VOC.LIST @  WHILE 1+ 6 = ABORT"  Too
many vocabularies" REPEAT VOC.LIST !
        \ Add to VOC.LIST
 DOES> SET.CONTEXT ;

(FIND) is a primitive which performs the function of FIND on
a single vocabulary, searching for a match until it finds a
dummy name.

: FIND CONTEXT @ SEARCH.VOC ! 6 1 DO SEARCH.VOC @ @ (FIND)
?DUP IF LEAVE ELSE SEARCH.VOC @ 2+ @ DUP 0= IF LEAVE THEN
SEARCH.VOC ! THEN LOOP ;

: NORMAL.SEARCH 6 2 DO I VOC.LIST @ DUP 0= IF DROP LEAVE
THEN NAME> 8 + ['] FORTH 6 + SWAP ! LOOP ;
     Makes FORTH follow each other vocabulary in search
order

: SEEK (---addr f) BL WORD COUNT HERE FIND ;

: SEARCHES NORMAL.SEARCH CONTEXT @ 2+ FORTH SEEK -1 = 0=
ABORT"  No such vocabulary" 6 + DUP CONTEXT ! SWAP ! ;
   Checking for -1 rather than simply a true flag avoids
   a hangup if <RETURN> is pressed with no vocabulary name
   entered


: SEARCH.ORDER CR CONTEXT @ BEGIN DUP 6 - >NAME ID. 2+ @ DUP
WHILE ." searches " REPEAT DROP ;

Used as

  <vname> SEARCH.ORDER

prints out the search order starting with <vname> and ending
with FORTH.
```

(Figures, continued on page 30)

Chester H. Page earned his doctorate in mathematical physics at Yale and spent some 36 years at the National Bureau of Standards. His first Forth was Washington Apple Pi's fig-FORTH, which he modified to use Apple DOS, then ProDOS, and later to meet the Forth-79 and Forth-83 Standards. Recently, he added many features of F83, including a four-thread dictionary (but no shadow screens) and a vocabulary name format that provides for a search-order routine.

THE CHALLENGE OF SORTS

The Forth Interest Group (FIG) is pleased to announce a challenge to all Forth programmers. Beat our sort program and have a chance to win a prize of your choice. The author of the program judged best in our tests will get to choose between free on-line access to the FIG RoundTable on GEnie for one month, a $150 credit toward purchases from the FIG Mail Order Form, or a check for $100.

The Rules

Submissions must be electronically transmitted to the Software Libraries in the FIG RoundTable on GEnie no later than midnight November 31, 1989. The results and the winning entry will be published in the March/April issue of *Forth Dimensions*. All entries and results will be available on most Forth Bulletin Board systems soon after testing is complete. All submissions become the property of the Forth Interest Group for distribution as it sees fit. The source code for all entries must comply with the Forth-83 Standard (published in 1983 by the Forth Standards Team), a document available on the FIG Mail Order Form. The source code may be submitted in text or block format, but must comply to the conventions in the block file SORT.BLK (see following). Submissions will be compiled and tested with this test suite, and the average score after `80 TESTS` will be used to compare it to other submissions. The examples included in SORT.BLK provide best- and worst-case examples for sorting algorithms. The `BUBBLE` sort is the simplest, and the `QUICK` sort is a modification (by Wil Baden) of a sorting algorithm developed by C. Hoare. Figure One gives a sample of the statistics generated by each on the judges' system.

Although we encourage you to beat the score of our `QUICK` sort, that is not necessary to win this competition. The winner will be chosen from the valid submissions, based on the lowest average score (the last entry in the right-hand column after `80 TESTS`). Submissions will be disqualified if they do not comply to the Forth-83 Standard or if they fail to execute under this test suite.

Test Details

Dictionary bytes are determined by the size of the submitted sort after being compiled into our version of Forth. This Forth is based on the popular F83 model developed by Harry Laxen and Mike Perry. Although it is upwardly compatible with F83, we do not guarantee that the entire test suite will run under your version. In addition, since each version of Forth differs in how it compiles source code, do not assume that you can duplicate our results.

Figure Two shows statistics about how our Forth compiles source code. It is not intended to be a complete list of how our version of F83 works, but should give you an indication of how it differs from the F83 model. Refer to *Inside F83* by C.H. Ting (see the FIG Mail Order Form) for more complete details.

RAM words are determined by memory usage outside the Forth dictionary; this includes the parameter and return stacks, `PAD`, `TIB`, and any other memory usage between these areas and the top of the dictionary (referenced by `HERE`). See the table cited above for indications of how this number is affected; I have found it extremely hard to calculate, and have noticed that it is high by about 26 items. However, it is sufficient for the sake of this test.

Fetches and *stores* are affected by access to the `DATA` array to be sorted. They are incremented by the words `S@` and `S!` which must be used for all accesses into the `DATA` array.

Although this test suite is only based on a sort of 1024 bytes, it would not be useful to limit a sorting algorithm to this size. It should be assumed that there could be an unlimited number of data items and that the data could be of any size. The sorting algorithm should be easily modifiable to accommodate any variations in the data format.

Compares are incremented by use of the word `COMPARE` which also must be used in your sorting algorithm. It will return a number that represents the difference between two data items, according to the following truth table:

```
n1 < n2 = -1
n1 = n2 = 0
n1 > n2 = 1
```

The execution *time* is based the MS-DOS time function call which returns the current time down to 1/100th of a second. Although it is generally accurate, it has shown variations of up to ±5/100ths of a second. This should not be significant, though, since the time is scaled by the number of bytes we are sorting and will only give us an error of ±5/102400ths in our final score. As best as possible, we have tried to isolate the execution time of the sort itself, but there is a slight overhead encountered that is not measurable on our test machine. The tests will be run on a 12.5 Mhz 80386 computer running MS-DOS version 3.21. The score is based on a calculation combining all the other numbers in the following formula:

```
((Fetches+Stores+Compares)+
((Dict+RAM)*Time)/100)/BYTES
```

This will weight the memory usage

based on the amount of time the sort takes to execute, and will scale everything by the number of bytes being sorted. Although this is a fairly arbitrary measure of efficiency, it makes a sort that minimizes data access come out with the lower *score*. Under normal conditions, this could be considered the goal of any sorting algorithm.

The *maximum* is the score based on the individual maximums of each of the above items. This will indicate a worst case for the sorting algorithm. However, it is highly unlikely that the results would ever be produced on any one test. This number will only be used to resolve a tie.

The *average* is the score based on the individual average of each of the above items. It should indicate how the sort will perform under a variety of situations. This is the number we will use as the basis of our comparison.

The Data

The `DATA` array contains `1024 ITEMS` to be treated as 16-bit signed values.

There are eight types of data patterns that we will cycle through during the tests. Each pattern will be used ten times during our test, and each will contribute to the scores:

- The `RAMP` is a simple array of ascending values. This array is already sorted, so it should produce the lowest score.
- The `SLOPE` is also a simple array, but of descending values. The values in the array need to be reversed.
- The `WILD` pattern contains random signed values in each element.
- The `SHUFFLE` pattern starts with the `RAMP`, then reorders each of the elements into a random pattern.
- The `BYTE` pattern consists of random eight-bit values. There will obviously be some duplication in this array.
- The `FLAT` array is filled with a single value. It will be a random value, but the array does not need to be rearranged.
- The `CHECKER` pattern consists of alternating values. Two random values are selected and placed into the even and odd addresses.
- The `HUMP` is a Gaussian distribution of values. This pattern has a bell shape when viewed in graphic format.

The Analysis

As described earlier, we selected a scoring system based on the criteria we consider important in a sorting algorithm. However, do not expect that you will be able to reproduce our exact results. To make timing comparisons before you submit your entry, base them on the results you

QUICK SORT

Test	Dict	RAM	Fetches	Stores	Compares	Time	Score	Maximum	Average
RAMP	400	50	9348	1023	7944	2.03	18.77	18.77	18.77
SLOPE	400	51	10383	2050	7951	2.26	20.89	20.89	19.83
WILD	400	46	17793	5881	11228	3.79	35.73	35.74	25.12
SHUFFLE	400	52	17823	5885	11253	3.79	35.81	35.81	27.80
BYTE	400	46	16317	5493	10201	3.46	32.76	35.81	28.79
FLAT	400	52	16255	7810	8064	3.51	32.91	37.69	29.47
CHECKER	400	51	16668	7530	8595	3.63	33.61	37.69	30.06
HUMP	400	43	15858	5575	9702	3.40	31.87	37.69	30.28

BUBBLE SORT

Test	Dict	RAM	Fetches	Stores	Compares	Time	Score	Maximum	Average
RAMP	52	40	1047552	0	523776	155.38	1548.45	1548.45	1548.45
SLOPE	52	43	2095104	1047552	523776	352.78	3613.22	3613.22	2580.57
WILD	52	43	1552494	504942	523776	250.57	2543.95	3613.22	2568.44
SHUFFLE	52	43	1542996	495444	523776	248.75	2525.23	3613.22	2557.58
BYTE	52	43	1240412	192860	523776	191.85	1928.96	3613.22	2431.82
FLAT	52	40	1047552	0	523776	155.77	1548.49	3613.22	2284.65
CHECKER	52	40	1048574	1022	523776	155.88	1550.49	3613.22	2179.81
HUMP	52	43	1163672	116120	523776	177.46	1777.75	3613.22	2129.53

Figure One. Sample statistics generated by the judges' quick-sort and bubble-sort routines.

Construct	Dictionary	RAM use
: Header	4 bytes	1 word
DO	4 bytes	6 words
LOOP	4 bytes	
IF	4 bytes	
ELSE	4 bytes	
UNTIL	4 bytes	
; etc.	2 bytes	
16-bit literal	4 bytes	1 word
32-bit literal	6 bytes	2 words

Figure Two. Examples of how the judges' Forth compiles source code.

obtain from running our examples on your computer. We will run the test 80 times, cycling through each data pattern ten times. We will upload the results from the last eight runs of each submission into the Bulletin Board section of the Forth RoundTable on GEnie, showing the individual scores for each data pattern. From there, they will be distributed to the other Forth Bulletin Board systems within our virtual network. The score based on the averages after the last run will be used to rank each entry. If there is a tie between two entries, we will use the score based on the maximums to break the tie. If there is still a tie, we will select the winner based on the readability of the source code and the documentation included with it. We will publish the three entries with the lowest scores in the March/April issue of *Forth Dimensions*.

All entries must either be uploaded to the Software Libraries of the Forth RoundTable on GEnie or mailed to the FIG business offices (P.O. Box 8231, San Jose, California 95155 U.S.A.), where they will be uploaded for you. All entries must contain the name, address, and telephone number of the author so that winners can be notified. The deadline for submissions is November 30, 1989. All submissions become the property of the Forth Interest Group.

May the best sort win!

SORT.BLK

[Also available for downloading from the GEnie Forth RoundTable.]

```
Screen   0
\ SORT.BLK  A Sorting Competition 11Jun89dar

( Consider this a challenge. The Forth Interest Group wants)
( to see how good you are.  Come up with a sort that will beat)
( this one and win your choice of valuable prizes.  Read the)
( documentation file that accompanies this source for complete)
( details about the prizes and rules for participation.)
( May the best sort win.   DaR   )

( As in all my code since 1986 the stack at the beginning of  )
( a line not starting with a control flow word, and at extra  )
( space in the middle of a line, is given by the most recent  )
( stack comment.  For Forth to be readable it is absolutely   )
( necessary that what is on the stack is known. After a)
( control flow word the stack is given by extra space or a    )
( stack comment.  WWB   )
```

```
Screen   1
\ Sort Comparison Utilities   11Jun89dar
DEFER SORT  ( # -- P: Sort the data )

FROM SORT.BLK   2  5 THRU  ( Data Access )

CREATE START( -- a P:Start of sort application code)

FROM SORT.BLK 6  8 THRU ( Sorting algorithm) HERE START -

CONSTANT DICTIONARY ( -- #  P:Bytes used for code )

FROM SORT.BLK   9 16 THRU  ( Testing routines )

: TESTS  ( # --   P: Run sort tests )   HEADER  0
DO ( ) I PATTERN  TEST-SORT  TEST-DATA RESULTS
LOOP;
```

```
Screen   2
\ Data Array and Utilities   11Jun89dar
:  CELLS ( a -- a'   P: Scale word size )     2* ;
: 2CELLS ( a -- a'   P: Scale double size )   2* 2* ;

1024 CONSTANT ITEMS  ( -- # P: Number of data items to sort)
CREATE DATA( -- a P: Data to be sorted ) ITEMS CELLS ALLOT

: D* ( dn dm -- dp    P: Double number multiply )
>R SWAP OVER ( n1 m1 n2 m1) * >R ( n1 m1)
  OVER >R    UM* ( dp ) R> R> SWAP R>
( dp n2*m1 n1 m2) * + + ;

: MU/NEAR (dn nd -- dq P:Double divide with rounding
DUP >R    MU/MOD   ( r dq ) >R >R   ( r)
2*    R@ 1 AND +     R> R> ROT   ( dq r)
R> > IF   ( dq ) 1 M+     THEN ;
```

```
Screen   3
\ Data Access Statistics    11Jun89dar
VARIABLE TIMES (-- a  P: Number of tests we have completed)

: !USE  ( a -- P: Increment usage counter)
 DUP 2@   ( a d )   1, D+  ROT 2! ;

: !MAX( a -- P: Store unsigned maximum) DUP >R 2@( d)
R@ 1 2CELLS + 2@    ( d0 d1 ) 2OVER 2OVER DU<
 IF  2SWAP  THEN  2DROP   ( d )   R> 1 2CELLS + 2! ;

: !AVG ( a -- P: Accumulate average) DUP >R 2@( d )
  R@ 2 2CELLS + 2@   ( d0 d2 )   TIMES @ S>D D*
  D+  ( d )  TIMES @ 1+ MU/NEAR  R> 2 2CELLS + 2! ;

: !RESULTS ( a --    P: Analyze )   DUP !AVG   !MAX ;
```

```
Screen   4
\ Data Access Utilities    11Jun89dar
2VARIABLE FETCHES( -- a P: Times fetched) 2 2CELLS ALLOT
2VARIABLE STORES(-- a P:Times stored )2 2CELLS ALLOT
2VARIABLE COMPARES(--a P:Times compared) 2 2CELLS ALLOT

: S@ ( # -- n   P: Must be used to fetch value )
CELLS DATA + @   ( n )   FETCHES !USE ;

: S! ( n # --    P: Must be used to store value )
CELLS DATA + !   (    )  STORES   !USE ;

:COMPARE( n1 n2 --1 | 0 | 1 P:Must be used for compares)
 2DUP < >R   > 1 AND   ( t )   R> OR  COMPARES !USE ;
```

```
Screen   5
\ Bubble Sort Example   11Jun89dar
: EXCHANGE ( #1 #2 -- P: Exchange items at indices)
 2DUP S@  SWAP S@  ROT S!  SWAP S! ;

: BUBBLE ( # --    P: Slow sort for comparison )
 1 DO  I 0 DO  J S@  I S@  COMPARE
 0< IF  I J EXCHANGE  THEN
 LOOP    LOOP ;

: .TIMER ( d1/100s --  P: Display timer in seconds )
 <#  # #  46 ( . ) HOLD  #S  #>
 ( a # )  8 OVER - SPACES  TYPE ;
```

```
Screen   6
\ Quick Sort Utilities  11Jun89dar
: ORDER-3
  ( f l -- f l # P: Order first, middle and last index)
  2DUP OVER - 2/   32767 AND + >R
DUP S@ R@ S@ COMPARE 0< IF  DUP R@ EXCHANGE THEN
 OVER S@ R@ S@ COMPARE  0> IF  OVER R@ EXCHANGE
 DUP  S@ R@ S@ COMPARE  0< IF  DUP  R@ EXCHANGE  THEN
 THEN  R> ;

:BOTH-ENDS ( f l p -- f' l' P: Trim ends) >R  ( f l )
  BEGIN  OVER S@ R@ COMPARE  0< WHILE  1 0 D+  REPEAT
  BEGIN  DUP  S@ R@ COMPARE  0> WHILE  1-  REPEAT
  R> DROP ;
```

```
Screen   7
\ Quick Sort List Processing   11Jun89dar
: PARTITION ( f l -- f l' f' l   P: Rearrange lists )
 ORDER-3 S@ >R 2DUP 1 -1 D+ ( f l f' l' )
 BEGIN  R@ BOTH-ENDS  2DUP 1+ U<
 IF  2DUP EXCHANGE  1 -1 D+
 THEN  2DUP SWAP U<
 UNTIL  R> DROP  SWAP ROT ;

: SINK ( f p # -- f   P: Do insertion )  ROT >R (p #)
  BEGIN  1-  2DUP S@ COMPARE  0<
  WHILE  DUP S@  OVER 1+ S!  DUP R@ =
  IF  S!  ( )  R> EXIT  THEN
  REPEAT  1+ S!  ( )  R> ;
```

```
Screen   8
\ Quick Sort Algorithm  11Jun89dar
: INSERTION ( f l --    P: Insertion sort )   2DUP U<
  IF  1+  OVER 1+ DO  ( f )  I S@ I SINK  LOOP DROP
   ELSE  2DROP  THEN ;

: HOARIFY ( f l -- ...P: Quick and Insertion sorts )
  BEGIN  2DUP 7 0 D+ U< WHILE  PARTITION ( f l' f' l)
  2DUP - >R  2OVER - R>  > IF  2SWAP  THEN
  REPEAT  INSERTION ;

: QUICK ( # -- P:Quick sort )   1-  0 SWAP  DEPTH >R
  BEGIN  ( ... ) HOARIFY DEPTH R@ < UNTIL R> DROP ;
' QUICK IS SORT

:SINKING ( n-- P:Insertion Sort)1- 0 SWAP INSERTION;
```



```
Screen   9
\ Random Number Generator  11Jun89dar
VARIABLE SEED ( -- a    P: Random data pattern )

: SETUP ( --   P:Setup random sequence) 1234 SEED ! ;

: RANDOM ( -- n    P: Calculate next random number )
 SEED @   ( n )  314159261 *  1+  DUP SEED ! ;

 :CHOOSE
 ( limit -- 0..limit-1 P:Choose next random in range)
RANDOM  ( limit n )  UM*  SWAP DROP ;

: GAUSS ( n -- u   P: Gaussian distribution )
 RANDOM 0   ( n d )  RANDOM 0 D+  RANDOM 0 D+
 RANDOM 0 D+  RANDOM 0 D+  RANDOM 0 D+
 6 UM/MOD SWAP DROP  UM* SWAP DROP ;
```

```
Screen  10
\ Random Data patterns   11Jun89dar
: RAMP(-- P :Ascending values)ITEMS 0DO I I S!LOOP ;
: SLOPE    ( -- P: Build sample of descending values )
ITEMS 0 DO  ITEMS 1- I -  I S!    LOOP ;
: WILD ( -- P: Build sample of random positive values)
 ITEMS 0 DO  RANDOM   I S!    LOOP ;
: SHUFFLE  ( -- P:Buildsample of shuffled sequence)RAMP
ITEMS 0 DO  ITEMS CHOOSE  I EXCHANGE  LOOP ;
: BYTE     ( --  P: Build sample of byte values )
  ITEMS 0 DO  256 CHOOSE   I S!  LOOP ;
: FLAT( -- P:Build sample of equal values)RANDOM ( n)
  ITEMS 0 DO  DUP   I S!  LOOP    DROP ;
:CHECKER( -- P: Checker board)RANDOM RANDOM (n1 n2)
  ITEMS 0 DO  DUP  I S!  SWAP  LOOP  2DROP ;
: HUMP     ( --    P: Gaussian or bell curved data )
ITEMS 0 DO  256 GAUSS  I S!    LOOP ;
```

```
Screen  11
\ Pattern Setup and Analysis   11Jun89dar
: PATTERNS ( --    P: Group data setup patterns )
  RAMP  SLOPE  WILD  SHUFFLE
  BYTE  FLAT  CHECKER  HUMP ;

: PATTERN(# --P: Set updata by test)DUP TIMES ! 8 MOD
  CELLS  ['] PATTERNS >BODY +   ( cfa ) PERFORM ;

: TEST-DATA(-- P:Check orderof data) DATA @ ITEMS 1
DO   ( prev )  DATA I CELLS + @  SWAP OVER >
 ABORT" Data has not been sorted"
 LOOP  DROP ;
```

```
Screen  12
\ Stack Usage Checks    11Jun89dar
2VARIABLESTACK( --a P: Sum of RAM usage)2 2CELLS ALLOT

HEX  A5A5A5A5CONSTANTMARK( --n P:Stack mark) DECIMAL

: FILL-RAM (-- P: Fill RAM with MARKers) MARK HERE !
  HERE DUP 1 CELLS +  RP@ OVER - CMOVE ;

: TEST-RAM ( --    P: Check RAM usage )
  0. STACK 2!  HERE 1+  1 CELLS NEGATE AND   ( a )
   BEGIN  DUP @  MARK - IF  STACK !USE  THEN
    1 CELLS +  RP0 @ OVER U< UNTIL  DROP
   STACK !RESULTS ;
```

```
Screen 13
\ Setup Sort Tests  11Jun89dar
2VARIABLE TIME ( -- a P: Sum of time)2 2CELLS ALLOT

: !TIME ( d1 d2 --   P: Store timing results )
 2SWAP D- TIME 2!  TIME !RESULTS ;

: TEST-SORT ( --   P: Test the sort algorithm )
 0. FETCHES 2!  0.STORES 2! 0.COMPARES 2!
 FILL-RAM  COUNTER  ( d )  ITEMS SORT
 COUNTER ( d1 d2 )  TEST-RAM  !TIME  ( )
 FETCHES !RESULTS  STORES !RESULTS
 COMPARES !RESULTS ;
```

```
Screen 14
\ Sort Test Reports    11Jun89dar
: HEADER ( --   P: Setup and display test header )
  FETCHES  3 2CELLS ERASE  STACK 3 2CELLS ERASE
 STORES   3 2CELLS ERASE  TIME  3 2CELLS ERASE
 COMPARES 3 2CELLS ERASE  SETUP  CR
  ." Test    Dict RAM Fetches  Stores Compares   "
 ." Time   Score Maximum Average" ;

: .RESULTS ( n --   P: Display results )
   >R  ( )  DICTIONARY  4 U.R
   STACK    R@ 2CELLS + 2@  4 UD.R
   FETCHES  R@ 2CELLS + 2@  8 UD.R
   STORES   R@ 2CELLS + 2@  8 UD.R
   COMPARES R@ 2CELLS + 2@  8 UD.R
   TIME     R> 2CELLS + 2@  .TIMER ;
```

```
Screen 15
\ Report test results    11Jun89dar
: .ANALYSIS ( n --   P: Calculate results )  >R  ( )
 FETCHES  R@ 2CELLS + 2@   ( dfetch )
 STORES  R@ 2CELLS + 2@ ( dfetch dstore )
  COMPARES R@ 2CELLS + 2@  ( dfetch dstore dcomp )
  D+ D+  100 ITEMS M*/ ( df+ds+dc/items )
  TIME     R@ 2CELLS + 2@   ( d dtime )
  STACK    R> 2CELLS + 2@ DROP  ( d dtime stack )
  DICTIONARY +  ITEMS M*/  ( d1 d2)  D+ .TIMER ;

: RESULTS ( -- P:Displaytest results) CR TIMES @( n)
 8 MOD CELLS  [ ALSO BUG ]  ['] PATTERNS >BODY + @
  ( cfa )  >NAME 8 L.ID  ( )  0 .RESULTS
  3 0 DO  I .ANALYSIS  LOOP  [ PREVIOUS ] ;
```

```
Screen 16
\ Random generator tests  11Jun89dar
VARIABLE CYCLE ( -- a P: Random cycle check ) 4 CYCLE !

: TALLY ( n -- P: Show n) BASE @ SWAP( [ base] n )
  36 BASE !  1 .R  ( [base] )  BASE ! ;

: TEST-RANDOM ( --   P: Test generator )
  PAGE  DATA ITEMS CELLS ERASE  ITEMS  ( k )  1 1
DO ITEMS CHOOSE ( k u)  DUP 64 /MOD AT CELLS DATA +
( k a )  DUP >R @  ( k tally )  DUP 0=
 IF  SWAP 1- SWAP  THEN
 1+ DUP TALLY  R> !  ( k )  DUP 0=
 IF  0 18 AT  I U.  LEAVE  THEN
 I CYCLE @ MOD  0= IF  PAGE  THEN
 LOOP  DROP ;
```

phrases they lead to. Personally, I can stand writing `TIME >` instead of `DETECTED`, etc., and it keeps the number of required words to a minimum—I mention them for purposes of discussion.

Choice of Time Units

Another point of ongoing discussion is the design/choice of the set of time units. Some people argue that milliseconds are all that is ever required (the code can easily be simplified to this end, if desired), but others feel that a multiplicity of time units is more complete and leads to more readable code.

My own feeling on this issue is that a variable for millisecond-of-minute (0–59,999 unsigned) and another variable for minute-of-week (or even minute-of-month) provides millisecond resolution over a period of more than 45 days in a standard Forth double number.

Also, in this design `NEW`, `LAPSE`, etc. could be run as infrequently as every 59.99 seconds and still provide exact millisecond calculation of elapsed times. In the end, this decision depends on the source of the timing information—if there is a real-time clock in the system, I usually implement whatever the hardware provides.

Dave Edwards is a qualified electronic engineer who formed Jarrah Computers—a microprocessor engineering consultancy using Forth as a key element—four years ago. His company has specialized in the design of custom microcontrollers, ranging from the 68705 single-chip family to large industrial systems based on Rockwell's 65F11 Forth chip and, recently, Motorola's 68HC11.

(Pages' figures, from page 23)

vname	link	code	set.context	dummy name	dummy link	dummy parm

Figure One. A dummy link and parameter make the vocabulary connections.

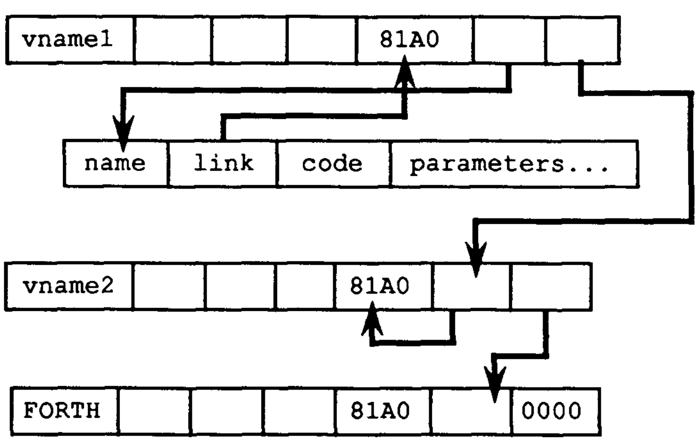


Figure Two.

THE BEST OF GENIE

GARY SMITH - LITTLE ROCK, ARKANSAS

In my rush to demonstrate how the GEnie Forth RoundTable was involved in the standards effort, how erudite and informative the guests in our real-time conferences are, and other impressive bits, I overlooked a facet that may be one of our most important services. This is how we stand as a resource center not only to the Forth expert, but also—perhaps even especially—to the new users of Forth.

I must begin with the Sunday night "Figgy Bar," usually conducted by Leonard Morgenstern. Leonard, and sometimes lead sysop Dennis Ruffer, conduct learning and technical sessions aimed at the new and intermediate Forth user. I have never come from these Sunday FIGGY's without some better understanding of Forth, so do not assume it is only for beginners. The point is, it is especially for beginners. No question is too trivial, so the first step to learning Forth the GEnie way is the Sunday night real-time conference.

Also, several files in the library can assist the newcomer. Browsing just the keyword "tutorial" generates an impressive list of files worth looking at, including Bill Kibler's Forth tutorial written in Forth. All one needs to do is load this file in any Forth-83-compatible system (the public-domain version of F83 for your computer is also waiting in the library!), invoke Kibler's program, and then learn Forth in Forth.

In the bulletin board area, we also have Category 15, Topic 1: Jack Brown's F-PC Forth tutorial. Jack has created the best on-line Forth tutorial I have ever seen. It is intended for use with Tom Zimmer's F-PC, a Forth for PCs and clones. It can be followed using other kernels, though, and a companion text file for F83 is in the library.

What if someone just has a question? There is lots and lots of help available on the GEnie Forth RoundTable bulletin board. Answers are quick to come from the GEnie sysops and other GEnie users, or via ForthNet, which ties us to several other Forth gurus. Topics such as "Which Public-Domain Kernel" (Category 1, Topic 7), "Basics of the Forth Language" (Category 2, Topic 1), and "for us beginners? HELP" (Category 2, Topic 5) are obviously in place to serve the new Forth user.

Some sample problems and responses follow:

Category 1, Topic 7, Message 1
From: Todd Natkin
Subj: F-PC, F83, MMS FORTH, etc.

A simple question: Is F-PC the "correct" implementation of Forth for me to be learning? Is it considered the most current of the public-domain implementations? I have looked over the material downloaded and ordered the technical reference manual from Dr. Ting, but do not have the time to review all the different versions of Forth and then pick the best one.

Where do you stand on this issue?

Category 1, Topic 7, Message 2
From: Jerry Shifrin
Subj: F-PC, F83, MMS FORTH, etc.

>A simple question: Is F-PC the "correct" implementation of Forth for me?

Like, politically correct? It's on the approved list, okay?

You can do useful work with most of the available Forths. F-PC is good in that it has numerous add-ons already available. OTOH, there may be too much material for some people to be comfortable. For a smaller implementation, check out Martin Tracy's ZenForth.

Category 1, Topic 7, Message 3
From: John Somerville
Subj: F-PC, F83, MMS FORTH, etc.

Todd, hope you don't mind me butting in, but I am a relative newcomer to Forth and computing, so my experience may be of interest to you. I have tried several PD Forths and came to the following conclusions:

Laxen & Perry's Forth (F83) is very good, particularly if you pick up Jack Brown's `VEDIT`. However, it does not have floating point, nor graphics. You can metacompile it to run on your hard drive.

Zimmer's Forth is nice, particularly since Jack Brown has put out floating point for it. I have not tried the floating point, since I have purchased a commercial package. However, I found it too large.

I tried ZenForth, but there were a few versions all packed together, and I really had trouble knowing what documentation referred to what.

UniForth has a demo package which didn't look too bad, but I didn't feel right about using it and I had no luck getting in touch with the company.

I purchased the commercial package (UR Forth) because I was tired of fishing around without documentation. Also, I started using versions which someone had altered and I frequently found myself lost in an undocumented morass. However, I think all the boards on this network have virgin copies of the F83 and F-PC systems.

If I had to choose a public-domain Forth now, I would go for F-PC because of the floating point; but F83 still has more appeal because of its compactness. If you choose either of these two, I recommend getting a hard copy of the source code and the user manuals available through FIG.
regards
NET/Mail: British Columbia Forth Board
Burnaby, BC 604-434-5886

Category 1, Topic 7, Message 4
From: GARY-S [Gary]

MM20, and MX80 are text-oriented Forth kernels written for CP/M machines, while F-PC is written for PCs and compatibles. I hope this helps resolve some of the confusion.

Category 1, Topic 7, Message 5
From: M.Hawley

I've been going Forth for over a year now. My recommendation is to start with F83 and Brodie's book, *Starting Forth*. The two complement each other nicely. Forget floating point. It is a bad habit which you should unlearn. Later, if you really need it, it can be added. However, I still haven't found any good reason to use floating point. If you have a PC compatible, move up to F-PC after you are comfortable with F83, the line editor, and blocks. I think it important to be exposed to these for a general understanding of Forth. At least at first, download only applications written for your particular version of Forth. Otherwise you will go nuts trying to supply the "missing word" which hangs your loading process. With F-PC, you will have the luxury of a screen editor and sequential files to work with. You will need the documentation from Dr. Ting. Enjoy!

By the way, when you get stuck on a problem, don't be shy. Post a message to this board and the experts here will pitch in to help. They helped me several times. Let us know how you're doing...
a recent beginner —meh

Topic 33
From: J.Ventola
Sub: neophyte needs F83 examples

This topic is for pointing us neophytes to *examples* in F83 of doing simple things like getting input from a user.

Category 1, Topic 33, Message 26
From: K.Smith10

Just going over these messages for the first time and noticed some questions I (finally) might help with... One of the handiest things I found with using F83 (MS-DOS) is that you can load a screen—a single screen—from another file while you are in the process of loading screens (blocks) from a different application... a good example of this is the EXTEND86.BLK load screen, which loads CPU8086.BLK screen 1 and UTILITY.BLK screen 1; each of these screens is a list of `LOAD` instructions for the screens within its file, and all of this can be redirected or cancelled or added to as needed. I do my development with an F83.COM version that has all the utilities I might need or want; then, when I've finished my application, I take its file and only load what it needs—usually not a screen editor or debugger or dumping, etc. The load screens act as a vector table pointing to what you want to use, without having to physically copy a screen into your application file (you do that at compile time in memory).

Which reminds me of something else that came up in the messages above, which is that Forth code is pretty portable—I know, I know, I've had some *real* fun trying it—but most code is gonna follow, or build on, accepted Forth fundamentals. If the original programmer was careful, you'll find most of the CPU- or system-specific code factored out from the general code (i.e., if you're going to write directly to screen memory and bypass the standard system calls, which words like `EMIT` are usually built on, that code will be off in its own screen grouped with supporting code, all of it building up to provide the whole application with generalized words like "print"—you could rewrite the low-level screen-memory codes to use your system addresses, etc., or simply make up "print" from general Forth output words like `EMIT`.

Long winded! You'd think I was a Fortran programmer!

Category 1, Topic 33, Message 27
From: K.Smith10

Thought I'd better split up these replies into separate notes. J. Ventola brought up implementing Pilot in Forth, but also mentioned that he'd found a cheap version available, so... but I bet some of the useful qualities of Pilot would be handy, at least as a module, within Forth. For a reference on Pilot, I remember an article in *Computer Language* magazine, the July and September 1986 issues, titled "Interpreter Design and Construction, formal language definition and initial coding in Pilot." In the article, the subject really is formal language definition, but the vehicle is to define Pilot. Not sure, don't remember how strictly Pilot is actually followed, but the article will provide ideas on how to go about implementing a language, as well as discussing the attributes of Pilot.

Category 1, Topic 33, Message 28
From: K.Smith10

Computer Language magazine has been a great and enjoyable resource for me over the years, but for lots of Forth reference I recommend *Dr. Dobb's Journal*. Martin Tracy's "4th Column" would be interesting to a new or old Forth programmer. Something that has helped me understand and use Forth better is to look at other languages (for which it is often easier to find a larger variety of subjects covered and, generally, more references), and also to look into more general aspects of computer programming. I've found that, as I've gotten into Forth, what I thought was a lack of understanding of Forth on my part turned out to be a lack of understanding of how something goes on inside my computer! I needed to see how interrupts work, even on a simple level, before I could resolve some file I/O problems I had, for example. Forth handled my needs quite well, once I knew what had to be done and how to go about it. On a recent project written in both Turbo BASIC and F83, I reduced the program file size by 30%, reduced execution time about 60%, and made the source instructions much clearer using the Forth system!

Algorithms! Get a nice, readable book on data structures. Try implementing some modules in a language more familiar to you (BASIC, possibly), then again in Forth. I'll bet there's a wealth of advice and suggestions to be had in this vein here on GEnie's Forth forum! Am I right everybody?

Category 1, Topic 33, Message 30
From: NMorgenstern [Leonard]

To K.Smith: Yea and verily! Your experience is that of many others. Mahlon Kelly commented a few weeks ago on one of our Figgy Bar sessions that computer languages were designed to give the user access to the computer but, more importantly, to protect the computer from the user. This is accomplished, of course, by limiting what the language can do. Forth is free of these restrictions. Mahlon teaches Forth, and students have told him that, for the first time, they understand the computer.

Helen Burke, a friend of mine who is a well-known metal sculptor, talks a lot about organic form in her art, meaning that the form grows naturally from the materials and the function. Forth is organic in this sense, growing from the microprocessor and operating system rather than from a preconceived set of rigid ideas about what a computer language should look like, á la Wirth, Kernigan, and others. Regrettable...

Category 2 Introduction to Forth, Topic 5
From: M.Silva (Forwarded)
Sub: for us beginners? Help

We beginners need a place to get our feet wet. I am somewhat of an accomplished programer in assembler, Fortran, COBOL, Pascal... but not Forth. Where do I get started?

Category 2, Topic 5, Message 106
From: C.Struyckenl

Very basic question: I am trying to get condensed mode out of my stargemini10x using F83. In screen 44 of utility.blk, I changed this:

```
: epson
control o emit ;
```

to this:

```
: starcond
control 15 emit ;
```

and also replaced the `noop` in the next line with `starcond`. I then loaded the screen.

This does not seem to work. I tried to see if `save-system f83.com` would make a difference... it did not. How do I get this to work? I also noticed that the whole screen got reloaded, resulting in many "already exists" notices. Does this mean Forth has now two identical compilations of each of these "already exists" words? If so, what is the correct way to load a word without reloading the whole screen?

Category 2, Topic 5, Message 107
From: NMorgenstern [Leonard]
To: C.Struyckenl

"I changed control o emit to .... control 15 emit..."

In F83, the word `CONTROL` gets the next word from the input stream and masks its first character back to five bits. Thus, control-o is the same as 15 (decimal), while control-15 would be 1.

On Epson printers and many others, 15 (control-o) should put you into condensed mode. But you have to send it to the printer. You should type `PRINTING ON` first.

"I got a lot of 'already exists' messages."

Forth will warn if you are redefining a word. It is a warning, not necessarily an error, because sometimes you want to redefine something. In your case, it was an error. You need to `FORGET` the words you have defined. Thus, `FORGET FOO` removes the word `FOO` and all words subsequently defined. F83 makes forgetting easy by a special word, `MARK`. The first thing I do before loading anything is type `MARK TODAY`. Then, if I type `TODAY`, it forgets everything after `TODAY`, but not the word `TODAY` itself. F83 "makes it easy to forget," as the old song goes.

It sounds to me as if you are making good progress. Please keep asking questions—others learn from the answers, as well as yourself. Also, if you can, attend the Sunday night round tables. They are specially aimed at beginners like yourself. Good luck!

Category 2, Topic 5, Message 108
C.Struyckenl
To: NMorgenstern

Thanks very much for your help and encouragement. I am just starting to work my way through chapter nine of *Starting Forth* (second edition), and things are becoming a lot more confusing. In the meantime, I still have not resolved the printer mystery. (I *do* have a condensed printout of all the blocks now by flicking the appropriate dip-switch on the printer.) This is what I have discovered so far: When the printer is not hardware-forced into condensed mode and I use the command `PRINTING-ON`, typing `15 EMIT` or `CONTROL o EMIT` will software-force the printer into condensed mode. But, after having changed `NOOP` to `EPSON` in the second line of screen 44, i.e.,

```
DEFER INITPR
' EPSON IS INIT-PR
```

the words `PRINTING-ON`, `SHOW`, and `LISTING` should set the printer in condensed mode by themselves (because they all use `INIT-PR` in their definition). I get the feeling they all are still refering to the old `init-pr`, before I changed it. Does F83 use a precompiled UTILITY.BLK, and does it just pretend it is loading the screens? When, as an experiment, I tried to

FORGET the original EPSON, I got a "Below Fence" message. When does one need to do a SAVE-SYSTEM F83.COM, and why and when does one need to metacompile? Do these things all have to do with the fact that F83 is working under DOS?

Category 2, Topic 5, Message 109
From: Pete Koziar
Subj: printer initialization

One important step you left out: F83 does not invoke EPSON when printing; it invokes a deferred word called INIT-PR, which is set up to be a NOOP. To use that printer control, type:

```
' epson is init-pr
```

before you try to print or list anything. If you then want a listing, just type listing any time after redirecting INIT-PR. If you just want to echo what is on the screen in condensed mode, you would need to say:

```
printing {
```

There is another word, by the way, known as PAGE. If your printer supports automatic form-feeds (most do, nowadays), you should also type:

```
' form-feed is page
```

I hope this helps!
Via Qwikmail 2.01 The Baltimore Sun

Category 2, Topic 5, Message 110
From: C.Struyckenl
To: Peter Koziak
Subject: printer initialization

Thanks for your response, Peter. I had already reset NOOP to Epson, but this did not make it work either. I finally figured that the words in UTILITY.BLK must be precompiled and that, therefore, the other words that use init-pr in their definitions are using the older init-pr that was set to NOOP. Does this make any sense? Without really knowing what I was doing, I re-metacompiled the system and now everything is working. It is still not completely clear what the metacompiling does and how it differs from save-system, but maybe the "under the hood" chapter in *Starting Forth* will make things a bit clearer.

Category 2, Topic 5, Message 111
From: Steve Palincsar
Subject: F83 utility.blk

It's been several years since I seriously looked at F83, but as I recall you are absolutely correct in your surmise that it uses a precompiled UTILITY.BLK. All the .BLK files supplied in the .ARC file are there for documentation and have already been incorporated in the F83.COM file. I don't recall any optional extension files in the Laxen &Perry package itself that you need to load. (There are, of course, hundreds if not thousands of extensions for F83 in the public domain.)

Category 2, Topic 5, Message 113
From: F.Sergeant

C.Struycken1, there is no need to redefine the word `EPSON` or to put the new word in UTILITY.BLK next to `EPSON`. There is no need to recompile your Forth system. Leave `EPSON` alone and define a brand-new word that will initialize *your* printer, which I gather is not an Epson. Call the new word `STARCOND`, as you suggested, or `GEMINI10X`, or whatever. Put it anywhere in an empty screen.

You would want code something like this:

```
HEX
: GEMINI10X ( -- )
  15 EMIT ;
```

(This word puts the Gemini printer into condensed mode.)

```
' GEMINI10X IS INIT-PR
```

(This re-vectors `INIT-PR` so it will use your brand-new definition when it does `SHOW`, etc., *without* having to recompile your Forth system.)

```
SAVE-SYSTEM F83G.COM
```

(The "G" is to remind you that this version will work with your Gemini printer.)

I hope this clears things up. —Frank

* * *

In the next "Best of GEnie" column, we will look at how ForthNet has grown since its first faltering steps a few months ago.

Many of the messages above were posted to GEnie via ForthNet, thanks in large meassure to the unflagging efforts of Jerry Shifrin, sysop of the East Coast Forth Board.

To suggest an interesting guest, please leave e-mail posted to GARY-S on GEnie (gars on Wetware and the Well), or mail me a note. I encourage anyone with a message to share to contact me via the above or through the offices of the Forth Interest Group.

REFERENCE SECTION

Forth Interest Group
The Forth Interest Group serves both expert and novice members with its network of chapters, *Forth Dimensions*, and conferences that regularly attract participants from around the world. For membership information, or to reserve advertising space, contact the administrative offices:

Forth Interest Group
P.O. Box 8231
San Jose, California 95155
408-277-0668

Board of Directors
Robert Reiling, President *(ret. director)*
Dennis Ruffer, Vice-President
John D. Hall, Treasurer
Terri Sutton, Secretary
Wil Baden
Jack Brown
Mike Elola
Robert L. Smith

Founding Directors
William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge
John James

In Recognition
Recognition is offered annually to a person who has made an outstanding contribution in support of Forth and the Forth Interest Group. The individual is nominated and selected by previous recipients of the "FIGGY." Each receives an engraved award, and is named on a plaque in the administrative offices.

1979 William Ragsdale
1980 Kim Harris
1981 Dave Kilbridge
1982 Roy Martens
1983 John D. Hall
1984 Robert Reiling
1985 Thea Martin
1986 C.H. Ting
1987 Marlin Ouverson
1988 Dennis Ruffer

ANS Forth
The following members of the ANS X3J14 Forth Standard Committee are available to personally carry your proposals and concerns to the committee. Please feel free to call or write to them directly:

Gary Betts
Unisyn
301 Main, penthouse #2
Longmont, CO 80501
303-924-9193

Mike Nemeth
CSC
10025 Locust St.
Glenndale, MD 20769
301-286-8313

Andrew Kobziar
NCR Medical Systems Group
950 Danby Rd.
Ithaca, NY 14850
607-273-5310

Elizabeth D. Rather
FORTH, Inc.
111 N. Sepulveda Blvd., suite 300
Manhattan Beach, CA 90266
213-372-8493

Charles Keane
Performance Packages, Inc.
515 Fourth Avenue
Watervleit, NY 12189-3703
518-274-4774

George Shaw
Shaw Laboratories
P.O. Box 3471
Hayward, CA 94540-3471
415-276-5953

David C. Petty
Digitel
125 Cambridge Park Dr.
Cambridge, MA 02140-2311
617-576-4600

Forth Instruction
Los Angeles—Introductory and intermediate three-day intensive courses in Forth programming are offered monthly by Laboratory Microsystems. These hands-on courses are designed for engineers and programmers who need to become proficient in Forth in the least amount of time. Telephone 213-306-7412.

On-Line Resources
To communicate with these systems, set your modem and communication software to 300/1200/2400 baud with eight bits, no parity, and one stop bit, unless noted otherwise. GEnie requires local echo.

GEnie
For information, call 800-638-9636
- Forth RoundTable *(ForthNet link*)*
  Call GEnie local node, then type M710 or FORTH
  SysOps: Dennis Ruffer (D.RUFFER), Scott Squires (S.W.SQUIRES), Leonard Morgenstern (NMORGENSTERN), Gary Smith (GARY-S)
- MACH2 RoundTable
  Type M450 or MACH2
  Palo Alto Shipping Company
  SysOp: Waymen Askey (D.MILEY)

(Continued on next page)

(Reference Section continued)

BIX (ByteNet)
For information, call 800-227-2983

- Forth Conference
  Access BIX via TymeNet, then type
  j forth
  Type FORTH at the : prompt
  SysOp: Phil Wasson (PWASSON)
- LMI Conference
  Type LMI at the : prompt
  Laboratory Microsystems products
  Host: Ray Duncan (RDUNCAN)

CompuServe
For information, call 800-848-8990

- Creative Solutions Conference
  Type !Go FORTH
  SysOps: Don Colburn, Zach Zachariah, Ward McFarland, Jon Bryan, Greg Guerin, John Baxter, John Jeppson
- Computer Language Magazine Conference
  Type !Go CLM
  SysOps: Jim Kyle, Jeff Brenton, Chip Rabinowitz, Regina Starr Ridley

Unix BBS's with Forth conferences (ForthNet links)*

- WELL Forth conference
  Access WELL via CompuserveNet or
  415-332-6106
  Fairwitness: Jack Woehr (jax)
- Wetware Forth conference
  415-753-5265
  Fairwitness: Gary Smith (gars)

PC Board BBS's devoted to Forth (ForthNet links)*

- East Coast Forth Board
  703-442-8695
  SysOp: Jerry Schifrin
- British Columbia Forth Board
  604-434-5886
  SysOp: Jack Brown
- Real-Time Control Forth Board
  303-278-0364
  SysOp: Jack Woehr
- Melbourne FIG Chapter
  Lance Collins
  (03) 299-1787 in Australia
  61-3-299-1787 international

(Letters, continued from page 6)

CASE will increment this count:

```
: +CASE ( -- )
  4 S>  4 S>
  1+
  4 >S  4 >S  ;

: CASE ( n -- )
  +CASE
  4 S@  = IF  ;
```

Finally, END will do all of the cleanup:

```
: FORCE ( -- )
  3 S> DROP
  1 3 >S  ;

: END ( -- )
  FORCE
  4 S> DROP
  4 S>
  0 DO THEN LOOP  ;
```

CASES is no longer needed, as its function has been absorbed by END.

With these changes, the case statement can help protect a programmer from an oversight or a miscount. The disadvantages here are some additional overhead in CASE and a larger case stack.

Enjoy,
Wes Cowley
P.O. Box 280138
Tampa, Florida 33682-0138
wcowley@dci2wc.das.net or
wes@cup.portal.com

On-line Down Under

Dear Editor,

The Melbourne Chapter of the Forth Interest Group wishes to acknowledge the support we have had in keeping our chapter going and in setting up our bulletin board.

We wish to thank Robert Reiling for his encouragement and help in obtaining an early copy of F-PC for us, and some other Forth software to start our board with last year.

We particularly thank Jerry Shifrin for his initial donation of files, which really gave our members something to think about. Recently, we have had another large batch of files from Jerry, which makes our board a major resource for Forth people here.

We want the Forth community to know that their efforts are greatly appreciated here.

Yours faithfully,
Lance Collins, Secretary
Melbourne Chapter

(Editorial, continued from page 4)

* * *

Some Forth notables are scheduled to appear at the Embedded Systems Conference in San Francisco on September 26–29. FORTH, Inc. will be joining a respectable exhibit floor with the likes of Advanced Micro Devices, H-P, Intel, and Tektronix. And Elizabeth Rather and Ray Duncan, along with P.J. Plauger and other pundits, will head intensive workshops during the event. This will be a fine opportunity for some cross-pollination, and it would be hard to find two better proponents of Forth to speak about embedded systems and real-time programming.

* * *

See your lawyer for details:

Some of our readers are consultants, at least part of the time, and some of them use consultants. A decision reached by the U.S. Supreme Court early this summer affects both groups by saying that freelance artists and consultants hold the copyright to all of their work unless a specific contract is made with their employer. This means that the consultant who writes that code might also own the rights to license and upgrade it.

The court's ruling may offer some protection to independent contractors, who often have little collection clout after they have turned in their work, but at the same time may make it scarier for companies to use them. Some fall into the habit of working without a written contract, but this decision provides motivation to put down in black and white exactly who is buying what from whom. It gives more reason than ever to be clear about work-for-hire and the distinctions between an employee and a consultant. (Source: *San Jose Business Journal* 7-17-89)

—*Marlin Ouverson*
Editor

FIG CHAPTERS

The FIG Chapters listed below are currently registered as active with regular meetings. If your chapter listing is missing or incorrect, please contact Kent Safford at the FIG office's Chapter Desk. This listing will be updated in each issue of *Forth Dimensions*. If you would like to begin a FIG Chapter in your area, write for a "Chapter Kit and Application." Forth Interest Group, **P.O. Box 8231, San Jose, California 95155**

U.S.A.

- **ALABAMA**
  Huntsville Chapter
  Tom Konantz
  (205) 881-6483

- **ALASKA**
  Kodiak Area Chapter
  Ric Shepard
  Box 1344
  Kodiak, Alaska 99615

- **ARIZONA**
  Phoenix Chapter
  4th Thurs., 7:30 p.m.
  Arizona State Univ.
  Memorial Union, 2nd floor
  Dennis L. Wilson
  (602) 381-1146

- **ARKANSAS**
  Central Arkansas Chapter
  Little Rock
  2nd Sat., 2 p.m. &
  4th Wed., 7 p.m.
  Jungkind Photo, 12th & Main
  Gary Smith (501) 227-7817

- **CALIFORNIA**
  Los Angeles Chapter
  4th Sat., 10 a.m.
  Hawthorne Public Library
  12700 S. Grevillea Ave.
  Phillip Wasson
  (213) 649-1428

  North Bay Chapter
  2nd Sat., 10 a.m. Forth, AI
  12 Noon Tutorial, 1 p.m. Forth
  South Berkeley Public Library
  George Shaw (415) 276-5953

  Orange County Chapter
  4th Wed., 7 p.m.
  Fullerton Savings
  Huntington Beach
  Noshir Jesung (714) 842-3032

  Sacramento Chapter
  4th Wed., 7 p.m.
  1708-59th St., Room A
  Tom Ghormley
  (916) 444-7775

  San Diego Chapter
  Thursdays, 12 Noon
  Guy Kelly (619) 454-1307

  Silicon Valley Chapter
  4th Sat., 10 a.m.
  H-P Cupertino
  Bob Barr (408) 435-1616

  Stockton Chapter
  Doug Dillon (209) 931-2448

- **COLORADO**
  Denver Chapter
  1st Mon., 7 p.m.
  Clifford King (303) 693-3413

- **CONNECTICUT**
  Central Connecticut Chapter
  Charles Krajewski
  (203) 344-9996

- **FLORIDA**
  Orlando Chapter
  Every other Wed., 8 p.m.
  Herman B. Gibson
  (305) 855-4790

  Southeast Florida Chapter
  Coconut Grove Area
  John Forsberg (305) 252-0108

  Tampa Bay Chapter
  1st Wed., 7:30 p.m.
  Terry McNay (813) 725-1245

- **GEORGIA**
  Atlanta Chapter
  3rd Tues., 6:30 p.m.
  Western Sizzlen, Doraville
  Nick Hennenfent
  (404) 393-3010

- **ILLINOIS**
  Cache Forth Chapter
  Oak Park
  Clyde W. Phillips, Jr.
  (312) 386-3147

  Central Illinois Chapter
  Champaign
  Robert Illyes (217) 359-6039

- **INDIANA**
  Fort Wayne Chapter
  2nd Tues., 7 p.m.
  I/P Univ. Campus, B71 Neff Hall
  Blair MacDermid
  (219) 749-2042

- **IOWA**
  Central Iowa FIG Chapter
  1st Tues., 7:30 p.m.
  Iowa State Univ., 214 Comp. Sci.
  Rodrick Eldridge
  (515) 294-5659

  Fairfield FIG Chapter
  4th Day, 8:15 p.m.
  Gurdy Leete (515) 472-7077

- **MARYLAND**
  MDFIG
  Michael Nemeth
  (301) 262-8140

- **MASSACHUSETTS**
  Boston Chapter
  3rd Wed., 7 p.m.
  Honeywell
  300 Concord, Billerica
  Gary Chanson (617) 527-7206

- **MICHIGAN**
  Detroit/Ann Arbor Area
  4th Thurs.
  Tom Chrapkiewicz
  (313) 322-7862

- **MINNESOTA**
  MNFIG Chapter
  Minneapolis
  Fred Olson
  (612) 588-9532

- **MISSOURI**
  Kansas City Chapter
  4th Tues., 7 p.m.
  Midwest Research Institute
  MAG Conference Center
  Linus Orth (913) 236-9189

  St. Louis Chapter
  1st Tues., 7 p.m.
  Thornhill Branch Library
  Robert Washam
  91 Weis Drive
  Ellisville, MO 63011

- **NEW JERSEY**
  New Jersey Chapter
  Rutgers Univ., Piscataway
  Nicholas Lordi
  (201) 338-9363

- **NEW MEXICO**
  Albuquerque Chapter
  1st Thurs., 7:30 p.m.
  Physics & Astronomy Bldg.
  Univ. of New Mexico
  Jon Bryan (505) 298-3292

- **NEW YORK**
  FIG, New York
  2nd Wed., 7:45 p.m.
  Manhattan
  Ron Martinez (212) 866-1157

  Rochester Chapter
  Odd month, 4th Sat., 1 p.m.
  Monroe Comm. College
  Bldg. 7, Rm.102
  Frank Lanzafame
  (716) 482-3398

- **OHIO**
  Cleveland Chapter
  4th Tues., 7 p.m.
  Chagrin Falls Library
  Gary Bergstrom
  (216) 247-2492

- **Columbus FIG Chapter**
  4th Tues.
  Kal-Kan Foods, Inc.
  5115 Fisher Road
  Terry Webb
  (614) 878-7241

  Dayton Chapter
  2nd Tues. & 4th Wed., 6:30 p.m.
  CFC. 11 W. Monument Ave. #612
  Gary Ganger (513) 849-1483

- **OREGON**
  Willamette Valley Chapter
  4th Tues., 7 p.m.
  Linn-Benton Comm. College
  Pann McCuaig (503) 752-5113

- **PENNSYLVANIA**
  Villanova Univ. Chapter
  1st Mon., 7:30 p.m.
  Villanova University
  Dennis Clark
  (215) 860-0700

- **TENNESSEE**
  East Tennessee Chapter
  Oak Ridge
  3rd Wed., 7 p.m.
  Sci. Appl. Int'l. Corp., 8th Fl.
  800 Oak Ridge Turnpike
  Richard Secrist
  (615) 483-7242

- **TEXAS**
  Austin Chapter
  Matt Lawrence
  PO Box 180409
  Austin, TX 78718

  Dallas Chapter
  4th Thurs., 7:30 p.m.
  Texas Instruments
  13500 N. Central Expwy.
  Semiconductor Cafeteria
  Conference Room A
  Clif Penn (214) 995-2361

  Houston Chapter
  3rd Mon., 7:30 p.m.
  Houston Area League of PC Users
  1200 Post Oak Rd.
  (Galleria area)
  Russell Harris
  (713) 461-1618

- **VERMONT**
  Vermont Chapter
  Vergennes
  3rd Mon., 7:30 p.m.
  Vergennes Union High School
  RM 210, Monkton Rd.
  Hal Clark (802) 453-4442

- **VIRGINIA**
  First Forth of Hampton Roads
  William Edmonds
  (804) 898-4099

  Potomac FIG
  D.C. & Northern Virginia
  1st Tues.
  Lee Recreation Center
  5722 Lee Hwy., Arlington
  Joseph Brown
  (703) 471-4409
  E. Coast Forth Board
  (703) 442-8695

  Richmond Forth Group
  2nd Wed., 7 p.m.
  154 Business School
  Univ. of Richmond
  Donald A. Full
  (804) 739-3623

- **WISCONSIN**
  Lake Superior Chapter
  2nd Fri., 7:30 p.m.
  1219 N. 21st St., Superior
  Allen Anway (715) 394-4061

INTERNATIONAL

- **AUSTRALIA**
  Melbourne Chapter
  1st Fri., 8 p.m.
  Lance Collins
  65 Martin Road
  Glen Iris, Victoria 3146
  03/29-2600
  BBS: 61 3 299 1787

  Sydney Chapter
  2nd Fri., 7 p.m.
  John Goodsell Bldg., RM LG19
  Univ. of New South Wales
  Peter Tregeagle
  10 Binda Rd.
  Yowie Bay 2228
  02/524-7490
  Usenet
  tedr@usage.csd.unsw.oz

- **BELGIUM**
  Belgium Chapter
  4th Wed., 8 p.m.
  Luk Van Loock
  Lariksdreff 20
  2120 Schoten
  03/658-6343

  Southern Belgium Chapter
  Jean-Marc Bertinchamps
  Rue N. Monnom, 2
  B-6290 Nalinnes
  071/213858

- **CANADA**
  BC FIG
  1st Thurs., 7:30 p.m.
  BCIT, 3700 Willingdon Ave.
  BBY, Rm. 1A-324
  Jack W. Brown (604) 596-9764
  BBS (604) 434-5886

  Northern Alberta Chapter
  4th Sat., 10a.m.-noon
  N. Alta. Inst. of Tech.
  Tony Van Muyden
  (403) 486-6666 (days)
  (403) 962-2203 (eves.)

  Southern Ontario Chapter
  Quarterly, 1st Sat., Mar., Jun., Sep., Dec., 2 p.m.
  Genl. Sci. Bldg., RM 212
  McMaster University
  Dr. N. Solntseff
  (416) 525-9140 x3443

  Toronto Chapter
  John Clark Smith
  PO Box 230, Station H
  Toronto, ON M4C 5J2

- **ENGLAND**
  Forth Interest Group-UK
  London
  1st Thurs., 7 p.m.
  Polytechnic of South Bank
  RM 408
  Borough Rd.
  D.J. Neale
  58 Woodland Way
  Morden, Surry SM4 4DS

- **FINLAND**
  FinFIG
  Janne Kotiranta
  Arkkitehdinkatu 38 c 39
  33720 Tampere
  +358-31-184246

- **HOLLAND**
  Holland Chapter
  Vic Van de Zande
  Finmark 7
  3831 JE Leusden

- **ITALY**
  FIG Italia
  Marco Tausel
  Via Gerolamo Forni 48
  20161 Milano
  02/435249

- **JAPAN**
  Japan Chapter
  Toshi Inoue
  Dept. of Mineral Dev. Eng.
  University of Tokyo
  7-3-1 Hongo, Bunkyo 113
  812-2111 x7073

- **NORWAY**
  Bergen Chapter
  Kjell Birger Faeraas,
  47-518-7784

- **REPUBLIC OF CHINA**
  R.O.C. Chapter
  Chin-Fu Liu
  5F, #10, Alley 5, Lane 107
  Fu-Hsin S. Rd. Sec. 1
  TaiPei, Taiwan 10639

- **SWEDEN**
  SweFIG
  Per Alm
  46/8-929631

- **SWITZERLAND**
  Swiss Chapter
  Max Hugelshofer
  Industrieberatung
  Ziberstrasse 6
  8152 Opfikon
  01 810 9289

SPECIAL GROUPS

- **NC4000 Users Group**
  John Carpenter
  1698 Villa St.
  Mountain View, CA 94041
  (415) 960-1256 (eves.)

Forth Interest Group
P.O.Box 8231
San Jose, CA 95155

Second Class
Postage Paid at
San Jose, CA